< **Sistem inteligent de dispensare a medicamentelor personalizat pentru afectiuni medicale** >

# Documentul de proiectare

Cuprins

[1. Introducere 1](#_Toc160527836)

[1.1 Scopul documentului 1](#_Toc160527837)

[2. Prezentare generală și abordări de proiectare 2](#_Toc160527838)

[2.1 Prezentare generală 2](#_Toc160527839)

[2.2 Presupuneri/ Constrângeri/ Riscuri 2](#_Toc160527840)

[2.2.1 Presupuneri 2](#_Toc160527841)

[2.2.2 Constrângeri 2](#_Toc160527842)

[2.2.3 Riscuri 3](#_Toc160527843)

[3. Considerațiii de proiectare 4](#_Toc160527844)

[3.1 Obiective și linii directoare (ghiduri) 4](#_Toc160527845)

[3.2 Metode de dezvoltare 4](#_Toc160527846)

[3.3 Strategii de arhitectură 4](#_Toc160527847)

[4. Arhitectura Sistemului și Proiectarea Arhitecturii 6](#_Toc160527848)

[4.1 Vedere logică 6](#_Toc160527849)

[4.2 Arhitectură hardware 6](#_Toc160527850)

[4.3 Arhitectură software 6](#_Toc160527851)

[4.4 Arhitectura informațiilor 7](#_Toc160527852)

[4.5 Arhitectura de comunicații interne 7](#_Toc160527853)

[4.6 Diagrama de arhitectură a sistemului 8](#_Toc160527854)

[5. Proiectarea sistemului 9](#_Toc160527855)

[5.1 Proiectarea bazei de date 9](#_Toc160527856)

[5.1.1 Obiecte de date și structuri de date rezultante 9](#_Toc160527857)

[5.1.2 Fișiere și baze de date 9](#_Toc160527858)

[5.2 Conversii de date 9](#_Toc160527859)

[5.3 Interfețe utilizator 10](#_Toc160527860)

[5.3.1 Intrări 10](#_Toc160527861)

[5.3.2 Ieșiri 10](#_Toc160527862)

[5.4 Proiectarea interfețelor cu utilizatorul 10](#_Toc160527863)

[6. Scenarii de utilizare 11](#_Toc160527864)

[7. Proiectare de detaliu 12](#_Toc160527865)

[7.1 Proiectare hardware de detaliu 12](#_Toc160527866)

[7.2 Proiectare software de deatliu 12](#_Toc160527867)

[7.3 Proiectare detaliată de securitate 13](#_Toc160527868)

[7.4 Proiectare de detaliu pentru performanța sistemului 14](#_Toc160527869)

[7.5 Proiectare detaliată a comunicațiilor interne (între componente) 14](#_Toc160527870)

[8. Controale pentru verificarea integrității sistemului 15](#_Toc160527871)

[Anexa A: Gestiunea modificărilor documentului 16](#_Toc160527872)

[Anexa B: Acronime 17](#_Toc160527873)

[Anexa C Documente la care se face referire 18](#_Toc160527874)

## Introducere

Documentul de Proiectare a Sistemului (DPS) se referă la proiectul intitulat **„Sistem automatizat de administrare și monitorizare a tratamentelor cu ajutorul unui microcontroler”**, care presupune realizarea unui sistem IoT compus dintr-un microcontroler (ESP32 sau Arduino Mega 2560), servomotoare SG90 și o aplicație mobilă pentru interacțiune cu utilizatorul.

Scopul acestui document este de a detalia modul în care cerințele specificate în documentul de cerințe (SRS) sunt transpuse în arhitectură și design tehnic. Acest document acoperă atât perspectiva de ansamblu (arhitectura sistemului), cât și specificațiile detaliate pentru fiecare componentă hardware și software.

Evoluția documentului se va realiza incremental, în paralel cu dezvoltarea sistemului, astfel încât fiecare modificare semnificativă a designului să fie reflectată corespunzător.

**Confidențialitate și securitate**: Acest document poate conține informații tehnice sensibile și trebuie utilizat doar de către echipa de proiect, dezvoltatori și alți membri autorizați. În cazul distribuției externe (ex: prezentări academice sau consultări), informațiile confidențiale vor fi anonimizate.

### Scopul documentului

Scopul acestui document este de a descrie structura, arhitectura și componentele tehnice ale sistemului ce urmează a fi implementat. El servește drept ghid pentru echipa de dezvoltare în fazele de proiectare, implementare, testare și mentenanță a aplicației și hardware-ului.

Acest document asigură coerența arhitecturii, urmărește deciziile de proiectare și permite trasabilitatea cerințelor funcționale în specificațiile de implementare.

Publicul țintă include:

* Managerul de proiect
* Echipa de dezvoltare (software și hardware)
* Utilizatori finali (pentru revizuirea interfeței)
* Profesori coordonatori (pentru evaluare academică)

## Prezentare generală și abordări de proiectare

Această secțiune descrie principiile și strategiile care vor fi utilizate ca ghiduri în momentul proiectării și implementării sistemului.

### Prezentare generală

Sistemul propus urmărește automatizarea procesului de eliberare a medicamentelor în funcție de un program prestabilit, configurabil printr-o aplicație mobilă. Microcontrolerul gestionează controlul servomotoarelor SG90 care deschid compartimentele cu medicamente, iar aplicația permite configurarea tratamentelor și oferă notificări de tip reminder.

Arhitectura este de tip distribuit, compusă din:

* **Partea hardware**: microcontroler (ESP32 sau Arduino Mega), servomotoare SG90, sursă de alimentare, senzori opționali.
* **Partea software**: aplicație mobilă (Android), backend Node.js, baza de date MongoDB, interfață de comunicare prin MQTT și HTTP/HTTPS.

Principiile directoare de proiectare includ modularitate, ușurință în utilizare, securitate și extensibilitate.

### Presupuneri/ Constrângeri/ Riscuri

#### Presupuneri

 Utilizatorii sistemului vor avea un smartphone compatibil cu aplicația (Android 8.0+).

 Microcontrolerul ales va avea conexiune stabilă Wi-Fi (în cazul ESP32) pentru comunicare cu serverul.

 Alimentarea electrică a dispozitivului va fi constantă, fără întreruperi frecvente.

 Utilizatorii vor introduce corect programul de tratament în aplicație.

 Sistemul backend va rula pe un server disponibil 24/7.

#### Constrângeri

 **Hardware**: Resurse limitate ale microcontrolerului (memorie, pini I/O).

 **Sistem de operare**: Aplicația mobilă va fi compatibilă doar cu Android (nu și iOS).

 **Comunicații**: Necesitatea unei conexiuni Wi-Fi stabile pentru funcționarea corectă a notificărilor și sincronizării cu serverul.

 **Interoperabilitate**: Se utilizează protocoale standard (HTTP, MQTT), dar integrarea cu alte sisteme smart home este limitată în prima versiune.

 **Securitate**: Transmiterea datelor se va face criptat prin TLS 1.2+, cu autentificare prin OAuth 2.0.

 **Testare**: Se vor realiza teste manuale, întrucât integrarea testării automate pentru hardware este limitată.

 **Limitări de memorie**: Pe ESP32 există constrângeri privind dimensiunea codului sursă și stocarea locală.

#### Riscuri

| **ID** | **Risc** | **Impact** | **Soluție propusă** |
| --- | --- | --- | --- |
| R1 | Pierderea conexiunii Wi-Fi | Mediu | Implementarea unui fallback pentru reîncercare la reconectare |
| R2 | Nealimentarea corectă a servomotoarelor | Ridicat | Utilizarea unei surse externe dedicate pentru motoare |
| R3 | Utilizatorul nu respectă notificările | Mediu | Adăugarea de notificări sonore și jurnale de activitate în aplicație |
| R4 | Atacuri asupra canalului de comunicare | Ridicat | Criptarea traficului (TLS), autentificare securizată (OAuth 2.0) |
| R5 | Aplicația nu este actualizată pe toate dispozitivele | Scăzut | Verificări automate pentru versiune minimă compatibilă |

## Considerații de proiectare

Instrucțiuni: Descrieți problemele care trebuie abordate sau rezolvate înainte de a încerca să elaborați o soluție de design completă.

### Obiective și linii directoare (ghiduri)

Sistemul este proiectat cu următoarele obiective și principii directoare:

* **Accesibilitate și ușurință în utilizare**: Interfața aplicației mobile este concepută pentru a fi intuitivă, cu butoane mari, clar etichetate și un flux logic simplu pentru utilizatori, în special vârstnici sau persoane cu experiență tehnologică limitată.  
  *Motiv: Publicul țintă include persoane care pot avea dificultăți în folosirea unor aplicații complexe.*
* **Fiabilitate și redundanță**: Sistemul trebuie să funcționeze constant și să fie capabil să gestioneze pierderile temporare de conexiune fără pierderi de date critice.  
  *Motiv: Medicamentele trebuie administrate la timp, iar sistemul nu trebuie să depindă exclusiv de conexiunea permanentă.*
* **Utilizarea eficientă a resurselor hardware**: Fiind bazat pe ESP32 (sau Arduino Mega), sistemul trebuie optimizat pentru memorie limitată și consum redus de energie.  
  *Motiv: ESP32 are memorie RAM limitată și este alimentat de obicei de la surse portabile.*
* **Respectarea convențiilor de programare și modularitate**: Codul este structurat pe module (funcții independente pentru controlul servo, conexiune Wi-Fi, comunicare MQTT etc.), respectând convențiile de denumire și comentarii clare.  
  *Motiv: Pentru mentenanță ușoară și scalabilitate.*
* **Compatibilitate și portabilitate**: Soluția este implementată în Arduino IDE folosind biblioteci standard și protocoale deschise (MQTT, HTTP), pentru a facilita portarea pe alte plăci sau sisteme.  
  *Motiv: Scalabilitate și adaptabilitate la alte proiecte similare.*

### Metode de dezvoltare

Pentru dezvoltarea sistemului hardware și software, s-a adoptat o **abordare incrementală bazată pe prototipare rapidă**, specifică proiectelor integrate cu microcontrolere.

* **Arduino IDE** a fost utilizat pentru dezvoltarea software-ului de control al hardware-ului (servomotoare SG90, senzori, conexiune Wi-Fi), folosind o abordare procedurală.
* Pentru partea mobilă, aplicația a fost dezvoltată utilizând **MIT App Inventor**, o platformă vizuală bazată pe blocuri logice, potrivită pentru prototipare rapidă și testare facilă pe dispozitive Android.
* Comunicația dintre module este testată iterativ cu pachete MQTT prin broker-ul **Mosquitto**, iar logica de program a fost verificată pe fiecare segment separat (control servo, trimitere date, afișare date în app).
* S-au explorat și alte opțiuni precum utilizarea unei aplicații în Android Studio, dar s-a optat pentru MIT App Inventor datorită timpului redus de dezvoltare și simplității integrării.

**Contingente posibile**:

* Instabilitatea conexiunii Wi-Fi sau incompatibilitatea rețelei locale a fost anticipată și s-a prevăzut un fallback prin salvarea locală temporară a comenzilor.
* În caz de schimbare a platformei hardware (de exemplu, comutare de la ESP32 la Arduino Mega fără conectivitate nativă), se poate adăuga un modul extern de comunicație (ex: ESP8266)

### Strategii de arhitectură

Arhitectura generală a sistemului este bazată pe o **topologie client-server asincronă**, unde:

* **ESP32** acționează ca **nod inteligent de control**, conectat la internet prin Wi-Fi și comunicând cu aplicația mobilă prin **protocolul MQTT**.
* **Aplicația mobilă** este interfața principală pentru utilizator, trimițând comenzi și primind notificări în timp real.
* **Servomotoarele SG90** sunt utilizate pentru controlul fizic al compartimentelor de medicamente, iar logica acestora este gestionată local de microcontroler.
* **Broker-ul MQTT (Mosquitto)** servește ca punct intermediar de comunicare și sincronizare a mesajelor.

**Raționamente și decizii**:

* Alegerea **ESP32** în detrimentul Arduino Mega se bazează pe faptul că ESP32 include nativ Wi-Fi, este mai rapid și mai eficient energetic.
* S-a optat pentru **protocolul MQTT** în locul HTTP deoarece este mai eficient în comunicații asincrone și permite gestionarea topicurilor într-un mod scalabil.
* Pentru ușurință în dezvoltare și testare, **MIT App Inventor** a fost preferat unei soluții avansate de tip Android Studio.
* În ceea ce privește detecția erorilor, s-au implementat funcții de reconectare automată și validare a mesajelor (ex: checksum, timestamp).
* Persistența datelor este gestionată pe termen scurt local în ESP32 (prin EEPROM, dacă e cazul), iar pe termen lung în cloud poate fi implementată extensia sistemului cu integrare Firebase sau Blynk.

## Arhitectura Sistemului și Proiectarea Arhitecturii

Sistemul este organizat în mod modular și se bazează pe o arhitectură distribuită client-server. Componenta hardware include un microcontroler (ESP32/Arduino Mega 2560), conectat la mai multe servomotoare SG90 și senzori, care gestionează în mod autonom distribuirea medicamentelor. Aplicația mobilă comunică prin Wi-Fi sau Bluetooth cu microcontrolerul, trimițând comenzi și primind date de stare. Datele relevante sunt stocate într-o bază de date (Firebase sau SQLite), accesibilă din aplicația mobilă.Această secțiune conturează designul arhitecturii sistemului și hardware-ului.

### Vedere logică

Vederea logică evidențiază împărțirea sistemului în următoarele componente principale:

* **Modulul de control al hardware-ului**: gestionează mișcările brațului robotic și răspunde comenzilor primite.
* **Modulul de comunicare**: gestionează schimbul de date între microcontroler și aplicația mobilă.
* **Aplicația mobilă**: oferă interfața cu utilizatorul și trimite/primește date către/de la microcontroler.
* **Modulul de gestionare a bazei de date**: permite stocarea și accesarea istoricului privind administrarea medicamentelor.

### Arhitectură hardware

Sistemul hardware este centralizat în jurul microcontrolerului:

* **Microcontroler**: ESP32 (sau Arduino Mega 2560), controlează servomotoarele și comunicarea cu aplicația mobilă.
* **Servomotoare SG90**: acționează mecanismul de livrare al medicamentelor.
* **Senzori (opțional)**: pot fi integrați pentru a detecta poziția sau confirmarea livrării.
* **Surse de alimentare**: asigură tensiunea necesară funcționării dispozitivelor.
* **Conectivitate**: ESP32 oferă conexiune Wi-Fi/Bluetooth pentru comunicare cu aplicația mobilă.

Diagrama hardware (de exemplu):  
ESP32 → SG90 Servo x N  
ESP32 ↔ Aplicație mobilă (Bluetooth/Wi-Fi)  
ESP32 → (opțional) Senzori

### Arhitectură software

 **Firmware (ESP32/Arduino)**: scris în C++ (platforma Arduino IDE), gestionează comenzile venite prin Wi-Fi/Bluetooth și controlează motoarele SG90.

 **Aplicația mobilă**: realizată în Android Studio (Java/Kotlin), cu interfață intuitivă pentru programarea și monitorizarea tratamentului.

 **Bază de date**: Firebase (realtime database) sau SQLite (stocare locală).

 **Biblioteci software**:

* Servo.h – pentru controlul motoarelor SG90.
* WiFi.h / BluetoothSerial.h – pentru conectivitate.
* FirebaseESP32.h – pentru interacțiunea cu baza de date Firebase.

 **Sisteme de operare**:

* Firmware – bare-metal (fără OS)
* Aplicația mobilă – Android

 **API/Protocol**:

* Protocoale de comunicație: Bluetooth SPP sau TCP/IP prin Wi-Fi.
* Structuri de date JSON pentru schimbul de informații între aplicație și microcontroler.

### Arhitectura informațiilor

 **Tipuri de informații stocate**:

* Profil utilizator (nume, vârstă, diagnostic)
* Programare medicamente (ora, tip, doză)
* Istoric administrare

 **Formatul datelor**:

* Date electronice (structuri JSON, tabele SQL)
* Introduse manual de utilizator prin aplicație

 **Furnizori de date**:

* Utilizatorul introduce datele inițiale.
* Microcontrolerul trimite feedback despre livrare (dacă este cazul).

 **Date sensibile**:

* Da, informațiile medicale sunt sensibile – este necesară criptarea și protecția accesului (parolă/cont).

### Arhitectura de comunicații interne

 **Tip de comunicație**: wireless

* Bluetooth (ESP32 ↔ telefon mobil) – conexiune locală directă
* Wi-Fi (ESP32 ↔ Firebase ↔ aplicație mobilă) – conexiune remote

 **Componente rețea**:

* ESP32 cu modul Wi-Fi/Bluetooth integrat
* Router Wi-Fi (dacă se folosește Firebase)

 **Flux de comunicație**:

* Utilizator → aplicație mobilă → comenzi → ESP32
* ESP32 → actualizări stare → aplicație mobilă/Firebase

 **Estimări resurse**:

* Transferuri mici de date (comenzi, stări) → trafic de rețea redus
* Cerințe minime pentru lățime de bandă (LAN/Wi-Fi)

### Diagrama de arhitectură a sistemului

## Proiectarea sistemului

### Proiectarea bazei de date

Instrucțiuni: Descrieți proiectarea tuturor fișierelor de sistem de gestionare a bazelor de date (DBMS) și a fișierelor non-DBMS asociate cu sistemul. Furnizați un dicționar de date cuprinzător care să conțină numele elementului de date, tipul, lungimea, sursa, regulile de validare, întreținerea (creare, citire, actualizare, ștergere (CRUD)), stocarea datelor, ieșirile, aliasurile și descrierea.

#### Obiecte de date și structuri de date rezultante

Obiectele de date principale includ:

* **Pacient**
* **Medicament**
* **Program Administrare**
* **Jurnal Evenimente**
* **Utilizator (administrator/pacient)**

{

"nume\_pacient": "Ion Popescu",

"medicament": "Paracetamol",

"ora\_administrare": "08:00",

"cantitate": "1",

"status": "Administrat"

}

#### Fișiere și baze de date

Sistemul folosește o bază de date Firebase Realtime (NoSQL), structurată ierarhic:

/users

/userId

- nume

- rol

- email

/medicamente

/medicamentId

- denumire

- doza

- frecventa

- ora

/sesiuni

/ziCurenta

- userId

- medicamentId

- status

- ora

| **Nume câmp** | **Tip** | **Lungime** | **CRUD** | **Regulă Validare** | **Descriere** |
| --- | --- | --- | --- | --- | --- |
| nume | String | 50 | C/R/U | Alfabetic, obligatoriu | Numele utilizatorului |
| email | String | 100 | C/R/U | Email valid, unic | Adresa de email |
| denumire | String | 100 | C/R/U | Alfabetic | Numele medicamentului |
| ora | String | 5 | C/R/U | Format HH:MM | Ora administrării |
| status | String | 10 | C/R | "Administrat"/"Omitere" | Starea administrării medicamentului |

##### Baze de date

Fișier JSON local pe ESP32 cu programările de administrare (backup):

* Tip: intrare-ieșire
* Actualizare zilnică (la sincronizare cu aplicația)
* Backup la fiecare 24h
* Dimensiune estimativă: <100 KB
* Utilizat de modulul principal ESP32 pentru compararea timpului curent cu programul

##### Fișiere non-DBMS

Sistemul utilizează câteva fișiere non-DBMS pentru configurări, loguri și schimb de date:

### ****1. config.json****

* **Tip**: JSON | **Utilizare**: Intrare/Ieșire
* **Modul**: Aplicație mobilă
* **Scop**: Stocare setări utilizator (nume, notificări, program dozare)
* **Structură**: Variabilă, max. 2 KB
* **Backup**: Local sau în cloud

### ****2. log.csv****

* **Tip**: CSV | **Utilizare**: Ieșire
* **Modul**: Aplicație + ESP32
* **Scop**: Istoric acțiuni (dozări, erori, modificări)
* **Structură**: timestamp, user, event, details
* **Dimensiune estimată**: ~30 KB/lună

### ****3. EEPROM ESP32****

* **Tip**: Memorie internă | **Utilizare**: Intrare/Ieșire
* **Modul**: ESP32
* **Scop**: Stocare ore dozare și stări sistem
* **Structură**: Fixă, max. 1 KB
* **Backup**: Se reface din aplicație

### ****4. buffer.txt (temporar)****

* **Tip**: Text | **Utilizare**: Temporar (BLE)
* **Modul**: ESP32 + aplicație
* **Scop**: Comunicare rapidă (comenzi/răspunsuri)
* **Structură**: Simplă, max. 1 KB

### Interfețe utilizator

### Clase utilizatori:

* **Administrator (aparținător/medic):**
  + Vizualizează și editează programul de administrare
  + Monitorizează statusul tratamentului
  + Estimați: 1-3 utilizatori simultan
* **Pacient:**
  + Primește notificări
  + Confirmă administrarea
  + Estimați: 1 utilizator simultan per dispozitiv

#### Intrări

* Interfață mobilă:
  + Selectare medicament din listă
  + Setare oră, frecvență
  + Confirmare administrare

**Validări:**

* Interval oră: 00:00 - 23:59
* Doză: numeric între 1-10
* Câmpuri obligatorii: toate

#### Ieșiri

**Rapoarte:**

* Istoric administrare (ultimele 7 zile)
* Status curent: „Administrat”/„Omitere”
* Afișare în aplicație + posibilitate export PDF

[Paracetamol] - Ora: 08:00 - Status: Administrat

[Nurofen] - Ora: 12:00 - Status: Neadministrat

### Proiectarea interfețelor cu utilizatorul

### Ecrane aplicație mobilă:

1. **Ecran principal:**
   * Lista medicamente programate
   * Buton de „Confirmă administrarea”
2. **Ecran setări program:**
   * Selectează pacient
   * Adaugă medicament + oră
3. **Ecran istoric:**
   * Afișare calendaristică administrări
   * Filtru după zi / medicament

Interfețele sunt realizate în Flutter/Dart, cu integrare Firebase pentru stocare și sincronizare.

## Scenarii de utilizare

Această secțiune descrie funcționalitatea generală a sistemului din perspectiva utilizatorilor, prin scenarii concrete ce evidențiază interacțiunea utilizatorilor cu componentele sistemului, în condiții operaționale normale și excepționale. Sistemul are două clase principale de utilizatori: **Pacientul** și **Administratorul (aparținător sau medic)**. Mai jos sunt prezentate scenarii reprezentative pentru fiecare categorie de utilizator.

### ****Scenariul 1 – Programarea administrării medicamentelor (Administrator)****

**Descriere:** Administratorul setează programul de administrare al unui pacient prin aplicația mobilă.

**Flux pas cu pas:**

1. Administratorul se autentifică în aplicația mobilă.
2. Selectează pacientul din listă.
3. Alege medicamentul din baza de date sau introduce unul nou.
4. Setează ora administrării, frecvența (zilnică, la 8h, etc.) și doza.
5. Aplicația salvează datele în Firebase și trimite un mesaj JSON către ESP32 prin rețea Wi-Fi.
6. ESP32 decodează mesajul și actualizează fișierul local de programare.
7. Sistemul confirmă administratorului că programarea a fost setată cu succes.

**Rezultat așteptat:** Programul de administrare este salvat și sincronizat corect între aplicație și dispozitiv.

### ****Scenariul 2 – Administrarea automată a medicamentului (Pacient)****

**Descriere:** La ora stabilită, sistemul activează compartimentul corespunzător prin SG90 pentru ca pacientul să primească medicamentul.

**Flux pas cu pas:**

1. ESP32 compară ora curentă cu programul salvat.
2. Când ora corespunde, servomotorul SG90 rotește clapeta compartimentului.
3. Pacientul este notificat prin buzzer/vibrație și mesaj în aplicație: „Este timpul să luați Paracetamol – 1 capsulă.”
4. Pacientul apasă butonul „Confirmă administrarea” în aplicație.
5. Sistemul înregistrează statusul ca „Administrat” în Firebase.

**Rezultat așteptat:** Medicamentul este oferit pacientului, iar confirmarea este salvată pentru raportare.

### ****Scenariul 3 – Omiterea unei administrări (Pacient)****

**Descriere:** Pacientul nu confirmă administrarea medicamentului în intervalul de timp prestabilit.

**Flux pas cu pas:**

1. ESP32 activează compartimentul la ora programată.
2. Nu se primește confirmare din partea pacientului în termen de 30 minute.
3. Sistemul înregistrează statusul ca „Neadministrat” și trimite alertă către administrator.
4. Administratorul primește notificare: „Pacientul Ion Popescu a omis administrarea Paracetamol la ora 08:00.”

**Rezultat așteptat:** Situația este înregistrată, iar administratorul este informat.

### ****Scenariul 4 – Actualizarea programului (Administrator)****

**Descriere:** Administratorul decide să modifice ora de administrare pentru un medicament.

**Flux pas cu pas:**

1. Se autentifică în aplicație și accesează secțiunea „Program administrare”.
2. Selectează medicamentul și modifică ora din 08:00 în 10:00.
3. Aplicația actualizează în Firebase, iar ESP32 este notificat imediat.
4. Programul local este modificat, iar notificările sunt reprogramate.

**Rezultat așteptat:** Programul este actualizat în timp real și fără pierderi de date.

### ****Scenariul 5 – Pierdere conexiune Wi-Fi (Situație excepțională)****

**Descriere:** Sistemul pierde conexiunea la rețea în timpul funcționării.

**Flux pas cu pas:**

1. ESP32 detectează lipsa conexiunii Wi-Fi.
2. Continuă să ruleze programul local din fișierul JSON salvat anterior.
3. Aplicația mobilă afișează mesaj de avertizare: „Dispozitiv offline.”
4. După restabilirea conexiunii, ESP32 sincronizează datele restante cu Firebase.

**Rezultat așteptat:** Sistemul funcționează autonom și sincronizează datele când conexiunea revine.

### ****Scenariul 6 – Generarea unui raport (Administrator)****

**Descriere:** Administratorul dorește să vizualizeze istoricul administrării pe ultimele 7 zile.

**Flux pas cu pas:**

1. Accesează aplicația și selectează secțiunea „Rapoarte”.
2. Alege pacientul și perioada dorită (ex. ultima săptămână).
3. Sistemul extrage datele din Firebase și afișează un tabel cu statusul fiecărui medicament.
4. Poate exporta raportul în format PDF.

**Rezultat așteptat:** Administratorul primește un raport clar și detaliat.

## Proiectare de detaliu

### Proiectare hardware de detaliu

Sistemul hardware este compus din următoarele componente principale:

| **Componentă** | **Model** | **Tensiune Alimentare** | **Interfață** | **Observații** |
| --- | --- | --- | --- | --- |
| Microcontroler | ESP32 DevKit | 3.3V / 5V | UART, Wi-Fi, BLE | Control principal, comunicare cu aplicația |
| Microcontroler | Arduino Mega 2560 | 5V | UART, I2C, GPIO | Control servomotoare, citire senzori |
| Servomotoare | SG90 (x6) | 5V | PWM | Control braț robotic dozator |
| Modul RTC | DS3231 | 3.3V / 5V | I2C | Timp real pentru programare dozare |
| Senzor IR | TCRT5000 | 5V | Digital | Detectare poziție în cutie |
| Ecran LCD | 16x2 cu I2C | 5V | I2C | Afișare mesaje/status |
| Butoane | -- | -- | Digital | Comenzi manuale locale |
| Alimentator | 5V 3A | 230V AC -> 5V DC | -- | Sursă generală de alimentare |

**Schema de conectare**: Include conexiunile pin cu pin (poate fi adăugată în Anexă o diagramă completă cu Fritzing sau similar).

**Specificații tehnice**:

* ESP32 are 520KB SRAM, dual-core CPU 240MHz.
* Arduino Mega are 256KB Flash, 8KB SRAM.
* SG90: unghi de rotație 0-180°, consum 500mA max în sarcină.
* Conectivitate: UART (ESP32 ↔ Arduino Mega), I2C (RTC, LCD), PWM (servo), GPIO (butoane, senzori).

### Proiectare software de detaliu

| **Identificator Serviciu** | **Clasificare** | **Definiție** |
| --- | --- | --- |
| ControlServos | Modul aplicație | Gestionează mișcarea brațului robotic folosind comenzile din aplicație sau programul prestabilit |
| SchedulerRTC | Serviciu de timp | Verifică ora actuală și declanșează dozele programate |
| BluetoothHandler | Comunicare | Gestionează conexiunea BLE cu aplicația mobilă |
| StateManager | Logică internă | Menține starea sistemului (idle, dozare, eroare) |
| UserInterface | UI embedded | Afișează informații pe LCD și interpretează comenzile de la butoane |

#### Exemplu detaliere pentru ControlServos:

* **Cerințe**: mișcare precisă, sincronizată, fără blocări.
* **Structuri de date interne**:

typedef struct {

int servoPin;

int currentAngle;

int targetAngle;

} ServoMotor;

 **Constrângeri**: max. 2 servomotoare active simultan (limitare curent).

 **Utilizatori/Interacțiuni**: apelează PWMManager, citește din StateManager.

 **Procesare**: verifică dacă un servo trebuie mișcat, trimite semnal PWM corespunzător, actualizează poziția.

### Proiectare detaliată de securitate

 **Autentificare**: prin aplicația mobilă, utilizatorul se loghează pe cont.

 **Autorizare**: doar utilizatorii logați pot modifica programul de dozare.

 **Jurnalizare și auditare**: aplicația salvează local log-uri cu dozele administrate.

 **Criptare**: conexiunea Bluetooth utilizează pairing securizat.

 **Porturi de rețea**: ESP32 folosește doar BLE (nu sunt expuneri TCP/UDP).

 **Detectare intruziuni**: sistemul afișează alertă dacă se detectează deconectarea unui servo sau eșec la dozare.

### Proiectare de detaliu pentru performanța sistemului

 **Cerințe capacitate**: stocare locală max. 10 doze programate; log 100 intrări.

 **Performanță**: acțiune completă (deschidere, livrare, închidere) sub 5 secunde.

 **Disponibilitate**: sistem activ 24/7; semnalizare în caz de eroare.

 **Fiabilitate**: dozare manuală disponibilă în caz de eșec software.

 **Backup & recuperare**: programul este salvat în EEPROM.

 **Puncte unice de eșec**: ESP32, alimentator – ambele sunt monitorizate periodic.

### Proiectare detaliată a comunicațiilor interne (între componente)

 **Număr de dispozitive**: 1 ESP32 (master), 1 Arduino Mega (slave), 6 servomotoare.

 **Sincronizare & control bus**: UART (9600 baud) între ESP32 și Mega, I2C 100kHz pentru senzori/LCD.

 **Format date**: JSON simplificat între aplicație și ESP32, comenzi de tip DOSE 2 între ESP32 și Mega.

 **Topologie**: ESP32 ↔ UART ↔ Arduino Mega ↔ PWM ↔ Servos.

 **Distanțe**: toate componentele sunt pe aceeași placă, sub 10cm între ele.

## Controale pentru verificarea integrității sistemului

### Securitate internă

* Accesul la aplicația mobilă este restricționat prin autentificare cu parolă/un cont de utilizator.
* Doar utilizatorii autorizați pot modifica programul de administrare a medicamentelor sau pot iniția dozarea manuală.
* Sistemul embedded ESP32 este programat să accepte doar comenzi de la aplicația asociată, utilizând un identificator unic și conexiune BLE securizată.
* EEPROM-ul este protejat: scrierea în memorie este permisă doar în moduri verificate (prin confirmare dublă din aplicație).

### Proceduri de audit

* Log-uri de sistem sunt generate de aplicație și salvează:
  + Timpul administrării fiecărei doze.
  + Utilizatorul care a inițiat acțiunea.
  + Mesajele de eroare sau alerte generate de ESP32 sau Arduino Mega.
* Rapoartele operaționale pot fi exportate lunar și păstrate timp de 6 luni.
* Aplicația permite generarea de fișiere CSV cu istoricul evenimentelor și log-uri de activitate.

### Piste de auditare a aplicațiilor

* Fiecare interacțiune cu datele critice (modificarea orei de dozare, anularea unei doze, schimbarea utilizatorului) este logată.
* Fiecare acces la datele programului de dozare este înregistrat cu identificatorul utilizatorului.
* Sistemul permite vizualizarea modificărilor efectuate într-o cronologie clară.

### Tabele standard pentru validarea câmpurilor de date

| **Câmp** | **Validare** |
| --- | --- |
| Ora dozei | Format 24h (HH:MM), interval valid |
| Nume utilizator | Minim 3 caractere, fără caractere speciale |
| Parolă | Min. 6 caractere, cel puțin o cifră |
| Număr compartiment | Între 1 și 6 |

### Procese de verificare pentru adăugare, ștergere, actualizare

* Orice modificare a programului de dozare solicită reconfirmarea parolei.
* Operațiile de adăugare și ștergere sunt logate cu dată și utilizator.
* ESP32 verifică integritatea datelor scrise în EEPROM printr-un checksum intern.
* Aplicația previne programarea simultană a mai multor doze la aceeași oră.

### Identificarea completă a informațiilor de auditare

Fiecare log conține următoarele date:

* ID utilizator
* Terminal de rețea (UUID telefon)
* Dată și oră (timestamp exact)
* Acțiunea efectuată (e.g., „doză administrată”, „program modificat”)
* Datele modificate (compartiment, oră, tip acțiune)

Anexa A: Gestiunea modificărilor documentului

Tabel 1 – Înregistrarea modificărilor asupreaa documentului curent

| versiune | Data | Autorul/Deținătorul | Descriere |
| --- | --- | --- | --- |
| 1.0 | 10.04.2025 | Tudoroiu Roxana | Versiune inițială a documentului de proiectare |
| 1.1 | 14.04.2025 | Tudoroiu Roxana | |  | | --- | |  |  |  | | --- | | Adăugare arhitectură detaliată hardware/software | |
| 1.2 | 16.04.2025 | Tudoroiu Roxana | Completare secțiunilor de audit, control și anexe |

Anexa B: Acronime

*.*

Tabel 2 - Acronime

| Acronim | Forma completă |
| --- | --- |
| BLE | Bluetooth Low Energy |
| ESP32 | |  | | --- | |  |  |  | | --- | | Microcontroler ESP32 (Embedded Systems Platform) | |
| EEPROM | |  | | --- | |  |  |  | | --- | | Electrically Erasable Programmable Read-Only Memory | |
| RTC | Real-Time Clock |
| PWM | Pulse Width Modulation |
| LCD | Liquid Crystal Display |
| JSON | |  | | --- | |  |  |  | | --- | | JavaScript Object Notation | |
| UART | |  | | --- | |  |  |  | | --- | | Universal Asynchronous Receiver-Transmitter | |

Anexa C Documente la care se face referire

Tabel 3 – Documente la care se facce referire

| Nume document | Locație sau URL | Dată emitere document |
| --- | --- | --- |
| Documentație ESP32 | https://www.espressif.com/en/products/socs/esp32 | 01.2023 |
| Foaie de catalog SG90 | https://datasheet.lcsc.com/lcsc/SG90.pdf | 03.2021 |
| Manual Arduino Mega 2560 | https://store.arduino.cc/products/arduino-mega-2560-rev3 | 12.2022 |
| Documentație Bluetooth Android API | https://developer.android.com/reference/android/bluetooth | 2024 |