

**UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” DIN IAŞI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

SPECIALIZAREA: INGINERIA SISTEMELOR

LUCRARE DE DIPLOMĂ

**Aplicație utilitară pentru asistarea
persoanelor cu probleme de mobilitate**

Absolvent:
Iacinschi Anda-Roxana

Coordonator științific:
Conf. dr. ing. Gabriela Varvara

Iași, 2025

CUPRINS

1 Fundamentarea teoretică și documentarea bibliografică	1
1.1 Contextul temei și domeniul de aplicare	1
1.1.1 Situația actuală a persoanelor cu probleme de mobilitate	1
1.1.2 Provocările persoanelor vulnerabile care locuiesc singure	2
1.2 Analiza realizărilor existente și comparații	4
1.2.1 Aplicații și platforme similare	4
1.2.2 Tehnologii utilizate în aplicațiile existente	5
1.3 Cerințele și specificațiile aplicației	6
1.3.1 Cerințe funcționale	6
1.3.2 Cerințe non-funcționale	7
1.3.3 Obiectivele soluției propuse	8
2 Proiectarea aplicației	9
2.1 Analiza platformei hardware	9
2.1.1 Senzori integrați	9
2.1.2 Conectivitate avansată	10
2.1.3 Putere de procesare	12
2.2 Modulele generale ale aplicației și interacțiunile	14
2.2.1 Descrierea modulelor principale	14
2.2.2 Fluxurile de date planificate	19
2.3 Proiectarea software	20
2.3.1 Arhitectura generală a aplicației	20
2.3.2 Diagrama ER a bazei de date	22
2.3.3 Tehnologiile utilizate	24
2.3.3.1 Aplicația mobilă	24
2.3.3.2 Backend	26
2.3.3.3 Platforma web	29
2.3.3.4 Baza de date	30
3 Implementarea aplicației și rezultatele obținute	33
3.1 Aplicația mobilă – Etape ale implementării	33
3.1.1 Inițializarea proiectului	33
3.1.1.1 Arhitectura aplicației mobile	33
3.1.1.2 2. Configurarea proiectului în Xcode	36
3.1.1.3 Biblioteci și framework-uri utilizate	36
3.1.1.4 Configurarea permisiunilor	36
3.1.1.5 Configurarea inițială a aplicației	37
3.1.1.6 Implementarea funcționalităților	38
3.2 Backend	51
3.2.1 Tehnologii utilizate	51
3.2.2 Structura backend-ului	52
3.2.3 Răspunsuri și validarea datelor	52

CUPRINS

3.2.4	Securitatea backend-ului	52
3.2.5	Interacțiunea cu aplicația mobilă	53
3.3	Platforma web – Etape ale implementării	53
3.3.1	Organizarea structurii proiectului	54
3.3.2	Configurarea mediului virtual	56
3.3.3	Configurarea backend-ului	58
3.3.4	Functionalități implementate	64
3.3.4.1	Fluxul de autentificare	64
3.3.4.2	Vizualizare și editare a datelor utilizatorilor	67
3.3.4.3	Gestionarea alertelor și vizualizarea pe hartă	70
4	Concluzii si implementari viitoare	75
4.1	Concluzii	75
4.2	Posibile Dezvoltări	76

Rezumat

Proiectul de față își propune dezvoltarea unei aplicații utilitare pentru asistarea persoanelor cu probleme de mobilitate care locuiesc singure, îmbinând o aplicație mobilă și o platformă web pentru a le oferi sprijin personalizat și asistență în situații de urgență. Scopul principal al acestei aplicații este de a utiliza senzorii telefonului mobil pentru a detecta și monitoriza schimbările brusăte de poziție ale utilizatorilor, precum și pentru a oferi sprijin prin notificarea unui dispecerat medical în timp real, 24/7. Prin intermediul tehnologiilor avansate integrate, proiectul contribuie semnificativ la creșterea siguranței și independenței acestor persoane vulnerabile.

Aplicația mobilă, dezvoltată nativ pe platforma iOS cu ajutorul limbajului Swift, permite utilizatorilor să se înregistreze și să se autentifice pentru a beneficia de funcționalități avansate. Aceasta utilizează senzorii telefonului, precum accelerometrul și giroscopul, pentru a monitoriza în timp real poziția și mișcările utilizatorilor. În cazul detectării unui impact, aplicația declanșează automat o serie de întrebări pentru evaluarea stării utilizatorului. Pe baza răspunsurilor, se generează alerte automate care sunt transmise dispeceratului medical, oferind posibilitatea coordonării eficiente a unei intervenții, cum ar fi trimitera unei ambulanțe în cazul în care utilizatorul nu poate răspunde sau se află într-o stare critică.

Platforma web completează ecosistemul, fiind dedicată atât pacienților, cât și personalului dispeceratului. Utilizatorii pacienți pot actualiza datele personale și pot comunica detalii relevante către dispecerat. În același timp, personalul medical beneficiază de un tablou de bord bine structurat, care afișează în timp real incidentele raportate și informațiile critice despre pacienți, facilitând coordonarea intervențiilor de urgență. Actualizarea informațiilor în timp real între aplicația mobilă și platformă asigură sincronizarea constantă a datelor.

Funcționalitățile principale ale sistemului includ detectarea automată a incidentelor, notificarea dispeceratului medical și gestionarea profilurilor utilizatorilor. Arhitectura backend, dezvoltată în Python, asigură comunicarea fluidă și securizată între aplicația mobilă și platforma web, utilizând protocoale avansate pentru protecția datelor și integritatea informațiilor.

Proiectul demonstrează cum tehnologiile moderne pot fi folosite pentru a răspunde unor nevoi stringente din domeniul sănătății, oferind o soluție tehnologică integrată care îmbunătățește calitatea vieții utilizatorilor și sporește nivelul de siguranță al acestora. Această abordare holistică evidențiază potentialul tehnologiei de a transforma modul în care persoanele cu mobilitate redusă primesc sprijin și asistență, stabilind un standard ridicat de inovație și utilitate practică.



Figure 1: *Logo Aplicație utilitară pentru asistare persoane cu probleme de mobilitate.*

Capitolul 1 Fundamentarea teoretică și documentarea bibliografică

Acest capitol oferă o privire de ansamblu asupra contextului temei și domeniului său de aplicare, subliniind provocările și nevoile persoanelor cu probleme de mobilitate. De asemenea, sunt analizate realizările existente în domeniul tehnologiilor de sănătate digitală și sunt identificate cerințele și specificațiile necesare pentru dezvoltarea aplicației propuse. Structura capitolului include analiza contextului actual, soluțiile existente, precum și obiectivele și cerințele tehnice ale soluției dezvoltate.

1.1 Contextul temei și domeniul de aplicare

Acest subcapitol oferă o analiză detaliată a contextului în care se înscrie tema lucrării, subliniind provocările cu care se confruntă persoanele cu mobilitate redusă. Sunt explorate atât impactul acestor provocări asupra calității vieții, cât și modul în care tehnologia poate contribui la rezolvarea acestor probleme.

1.1.1 Situația actuală a persoanelor cu probleme de mobilitate

Persoanele cu mobilitate redusă reprezintă un segment vulnerabil al populației globale, confruntându-se zilnic cu provocări care le afectează profund calitatea vietii. Conform Organizației Mondiale a Sănătății (OMS), aproximativ 15% din populația globală trăiește cu o formă de dizabilitate, iar între 2% și 4% dintre aceștia prezintă dizabilități severe, care le limitează semnificativ capacitatea de a se deplasa sau de a desfășura activități esențiale zilnice fără ajutor [1]. Într-o societate aflată în plină dezvoltare tehnologică, aceste persoane rămân adesea marginalizate, confruntându-se atât cu bariere fizice, cât și cu provocări sociale și economice.

La nivel european, infrastructura publică și accesul la servicii esențiale pentru persoanele cu mobilitate redusă sunt insuficiente, în special în regiunile est-europene. Potrivit Eurostat, persoanele cu dizabilități sunt de trei ori mai predispuse să întâmpine dificultăți în utilizarea transportului public și accesarea spațiilor publice comparativ cu restul populației [2]. Această situație contribuie la izolare socială și limitează integrarea acestor persoane în activitățile comunitare.

În România, situația este mai dificilă, în special din cauza deficiențelor de infrastructură. Conform unui raport al Consiliului Național pentru Combaterea Discriminării (CNCD), mai puțin de jumătate dintre instituțiile publice sunt adaptate pentru persoanele cu dizabilități, iar majoritatea zonelor rurale nu oferă acces la infrastructuri adecvate [3]. Lipsa rampelor de acces, a trotuarelor adaptate și a transportului public accesibil creează un mediu ostil pentru aceste persoane, limitându-le accesul la educație, locuri de muncă și servicii medicale.

Dincolo de dificultățile fizice, persoanele cu mobilitate redusă se confruntă și cu provocări psihologice și sociale. Izolare socială și sentimentul de inutilitate sunt frecvente, alimentate de lipsa interacțiunilor zilnice și de dificultatea de a participa activ la viața comunității. Aceste provocări contribuie la o incidentă ridicată a problemelor psihologice, cum ar fi depresia și anxietatea, care afectează semnificativ calitatea vieții acestui segment al populației.

Această situație evidențiază necesitatea unor soluții integrate care să abordeze atât provocările

fizice, cât și cele sociale. Tehnologiile moderne oferă oportunități unice pentru a sprijini aceste persoane, iar dezvoltarea unei aplicații utilitare care să monitorizeze activitățile, să detecteze risurile și să faciliteze comunicarea cu apartinătorii reprezintă un pas important către o societate mai incluzivă și mai accesibilă.

Rolul tehnologiei în sănătatea digitală și telemedicina

Tehnologia joacă un rol esențial în transformarea serviciilor medicale, oferind soluții care îmbunătățesc accesul la îngrijiri și calitatea vieții pacienților. Sănătatea digitală, definită ca utilizarea tehnologiilor informaționale și de comunicații pentru optimizarea serviciilor de sănătate, a câștigat o importanță semnificativă în ultimii ani. În centrul acestor progrese se află telemedicina, care permite pacienților să acceseze consultații de la distanță, eliminând barierele fizice și geografice [4].

Contribuția sănătății digitale

Dispozitivele mobile, precum Apple Watch, utilizează senzori avansați, cum ar fi accelerometre și giroscopale, pentru a detecta modificări brusăte de poziție sau incidente critice, cum ar fi căderile. De exemplu, funcționalitatea de detectare a căderilor de pe Apple Watch trimit alerte automate către apartinători sau servicii de urgență dacă utilizatorul nu răspunde în timp util. Acest nivel de monitorizare reduce semnificativ dependența de îngrijitori permanenți și crește sentimentul de siguranță al utilizatorilor.

În plus, funcționalitățile de monitorizare a ritmului cardiac ale dispozitivelor inteligente permit măsurarea frecvență a pulsului și pot trimite notificări atunci când sunt detectate valori anormale. Aceste funcții sunt esențiale pentru persoanele care necesită supraveghere constantă din cauza dizabilităților sau a afecțiunilor cronice.

Telemedicina și beneficiile sale

Telemedicina reprezintă un pas important în îmbunătățirea accesului la îngrijiri medicale pentru persoanele cu mobilitate redusă. Platforme precum Teladoc Health facilitează consultațiile medicale video, gestionarea dosarelor și prescrierea rețetelor digitale [5]. Aceste soluții reduc timpul de așteptare și economisesc resurse, facilitând accesul la specialiști.

Tehnologiile emergente și impactul social

Integrarea inteligenței artificiale în sănătatea digitală deschide noi oportunități. Algoritmii IA permit analiza predictivă a datelor medicale, ajutând la identificarea risurilor înainte ca acestea să devină critice [6]. Totodată, aplicațiile mobile și platformele de comunicare reduc izolarea socială, oferind utilizatorilor oportunități de conectare cu familia, prietenii sau grupuri de suport.

1.1.2 Provocările persoanelor vulnerabile care locuiesc singure

Locuirea individuală este una dintre cele mai mari provocări pentru persoanele cu mobilitate redusă, acestea confruntându-se cu riscuri și dificultăți semnificative. Lipsa unei rețele de sprijin, combinată cu limitările fizice, contribuie la un mediu în care siguranța și sănătatea sunt frecvent compromise. În acest context, soluțiile integrate care vizează monitorizarea și sprijinul personalizat devin esențiale.

Riscuri fizice și medicale

Persoanele cu mobilitate redusă care locuiesc singure sunt deosebit de vulnerabile la accidente domestice, cum ar fi căderile. Conform Organizației Mondiale a Sănătății (OMS), căderile sunt una dintre principalele cauze de rănire severă la persoanele în vîrstă, generând milioane de spitalizări anual la nivel global [7]. În lipsa unei intervenții rapide, aceste incidente pot duce la complicații medicale severe, inclusiv fracturi, traume craniene și pierderea completă a autonomiei.

Un alt risc major este reprezentat de timpul de răspuns întârziat în cazul situațiilor de urgență. Pentru o persoană care locuiește singură, incapacitatea de a solicita ajutor imediat poate avea consecințe critice asupra stării de sănătate. Acest aspect subliniază necesitatea unor soluții tehnologice care să monitorizeze continuu activitățile zilnice și să trimită alerte automate în caz de urgență.

Impactul social și psihologic

Dincolo de riscurile fizice, izolarea socială este una dintre cele mai mari provocări cu care se confruntă persoanele vulnerabile care locuiesc singure. Lipsa interacțiunii sociale regulate contribuie la apariția depresiei, anxietății și chiar a unui declin cognitiv accelerat. Studiile arată că persoanele care trăiesc în izolare sunt de două ori mai predispuse să dezvolte afectiuni mentale comparativ cu cei care beneficiază de sprijin social regulat [8].

Aceste dificultăți sunt amplificate de sentimentul de inseguritate și de lipsa conexiunilor emotionale, ceea ce afectează profund calitatea vietii. Crearea unor rețele digitale de suport și integrarea comunităților virtuale pot reprezenta pași importanți în reducerea impactului negativ al izolării sociale.

Soluții tehnologice adaptate nevoilor persoanelor vulnerabile

Pentru a răspunde acestor provocări, soluțiile tehnologice integrate oferă perspective promițătoare. Dispozitivele portabile, cum ar fi brățările de sănătate și ceasurile inteligente, pot detecta automat căderile și să transmită alerte în timp real către dispecerate medicale sau membri ai familiei [9]. De asemenea, locuințele inteligente echipate cu senzori IoT (Internet of Things) și algoritmi de inteligență artificială pot monitoriza activitățile zilnice, detectând abateri semnificative de la rutină și generând notificări automate.

Un exemplu practic este integrarea senzorilor de mișcare cu camere video, care pot evalua activitatea utilizatorului în timp real. Aceste tehnologii oferă un sentiment sporit de siguranță și independentă pentru persoanele vulnerabile. În plus, aplicațiile mobile de telemedicină permit efectuarea consultațiilor regulate și verificarea stării de sănătate, reducând necesitatea deplasărilor frecvente la unitățile medicale.

Importanța integrării soluțiilor tehnologice

Prin utilizarea tehnologiilor moderne, locuirea individuală devine mai sigură și mai accesibilă pentru persoanele cu mobilitate redusă. Dezvoltarea unor sisteme de monitorizare continuă, alerte automate și interacțiuni digitale personalizate contribuie la reducerea riscurilor, creșterea autonomiei și îmbunătățirea calității vietii. Cu toate acestea, adoptarea acestor soluții trebuie să fie însoțită de politici publice și programe educaționale care să faciliteze accesul la tehnologie, în special în regiunile unde resursele sunt limitate.

1.2 Analiza realizărilor existente și comparații

1.2.1 Aplicații și platforme similare

În domeniul sănătății digitale, au fost dezvoltate diverse aplicații și platforme care facilitează monitorizarea sănătății și gestionarea situațiilor de urgență. Aceste soluții oferă un punct de plecare valoros pentru dezvoltarea unor aplicații adaptate nevoilor specifice ale persoanelor cu mobilitate redusă. Totuși, majoritatea platformelor existente se concentrează fie pe monitorizare generală, fie pe telemedicină, fără a integra complet funcționalități de detectare automată a incidentelor și alertare rapidă către un dispecerat medical.

Exemple de aplicații existente

Teladoc Health

Teladoc Health este o platformă globală de telemedicină care oferă consultații video în timp real, gestionarea dosarelor medicale și prescrierea rețetelor digitale. Deși oferă un model eficient de conectare între pacient și medic, Teladoc nu include funcționalități pentru detectarea automată a urgențelor sau monitorizarea continuă a utilizatorilor. Acest aspect limitează aplicabilitatea sa pentru persoanele cu dizabilități motorii, care ar beneficia de o monitorizare activă [5].

Life Alert

Life Alert este un sistem simplu, axat pe detectarea căderilor și alertarea automată a unui dispecerat. Sistemul include dispozitive portabile care detectează incidentele și trimit notificări automate. Cu toate acestea, tehnologia utilizată este relativ simplă, fără capacitate de monitorizare continuă a stării utilizatorului sau de analiză detaliată a datelor medicale [10].

Guardian Connect (Medtronic)

Guardian Connect este o aplicație specializată în monitorizarea continuă a nivelului de glucoză la pacienții cu diabet. Alertele automate generate în cazul valorilor critice evidențiază importanța conectivității în timp real între pacient și profesioniștii din domeniul sănătății. Totuși, aplicația este limitată la o anumită categorie de utilizatori și nu poate fi extinsă pentru gestionarea altor tipuri de urgențe medicale [11].

Analiza comparativă a funcționalităților

Aplicație/Platformă	Funcționalități cheie	Limitări
Teladoc Health	Consultații video în timp real, gestionarea dosarelor medicale.	Lipsa monitorizării automate sau a detecției de urgențe.
Life Alert	Detectarea căderilor, alertarea automată a unui dispecerat.	Tehnologie simplă, fără monitorizare activă continuă.
Guardian Connect	Monitorizare continuă, alerte pentru valori critice, conectivitate în timp real.	Specific pentru pacienți cu diabet, nu pentru alte afecțiuni.

Table 1.1: Analiza comparativă a aplicațiilor existente

Relevanța pentru aplicația propusă

Aplicația propusă în această lucrare își dorește să depășească limitările soluțiilor existente prin integrarea funcționalităților cheie:

- Detectarea automată a incidentelor.
- Alertarea rapidă și automată a unui dispecerat medical, asigurând un timp de răspuns minim.

Aceste funcționalități oferă o soluție mai completă și mai adaptată nevoilor persoanelor vulnerabile, punând accent pe siguranță și accesibilitate.

1.2.2 Tehnologii utilizate în aplicațiile existente

Aplicațiile pentru sănătate digitală utilizează o varietate de tehnologii avansate pentru a oferi funcționalități complexe, precum monitorizarea continuă, detectarea incidentelor și notificările automate. În această secțiune, sunt analizate două componente esențiale ale aplicațiilor existente: utilizarea senzorilor în dispozitivele mobile și arhitecturile software care permit integrarea și gestionarea acestor funcționalități.

Utilizarea senzorilor în dispozitivele mobile

Senzorii integrați în dispozitivele mobile joacă un rol crucial în monitorizarea sănătății utilizatorilor. Accelerometrele, giroscopul, senzorii de ritm cardiac și cei de saturăție a oxigenului din sânge sunt utilizati pe scară largă pentru a colecta date în timp real. Aceste date sunt procesate pentru a detecta modificări bruse ale poziției utilizatorului, ritmului cardiac sau altor parametri relevanți pentru sănătate.

De exemplu, *Apple Watch* folosește un accelerometru avansat și un giroscop pentru a detecta căderile și a genera notificări automate către contacte de urgență [12]. Similar, dispozitivele *Fitbit* monitorizează activitatea fizică, ritmul cardiac și nivelurile de stres, utilizând senzori avansați pentru a oferi feedback personalizat [13]. Acești senzori permit detectarea nu doar a activităților normale, ci și a evenimentelor critice, cum ar fi căderile sau nivelurile anormale ale parametrilor vitali.

În plus, integrarea senzorilor cu aplicațiile mobile permite utilizatorilor să vizualizeze și să analizeze datele colectate, oferindu-le o mai bună înțelegere a stării lor de sănătate. Această conectivitate este posibilă prin tehnologii precum Bluetooth și Wi-Fi, care asigură o transmisie rapidă și sigură a datelor.

Arhitecturi software pentru monitorizare și notificări

Pentru a sprijini funcționalitățile oferite de senzori, aplicațiile pentru sănătate digitală se bazează pe arhitecturi software complexe care integrează colectarea, procesarea și transmiterea datelor. Aceste arhitecturi sunt esențiale pentru a asigura că datele sunt analizate în timp real și că notificările sunt trimise rapid în caz de urgență.

De exemplu, *Teladoc Health* folosește o arhitectură bazată pe cloud pentru a stoca și procesa datele pacientilor [5]. Acest model permite accesul rapid și securizat la dosarele medicale și gestionarea eficientă a consultațiilor video. În mod similar, aplicațiile de monitorizare, precum *Life Alert*, utilizează algoritmi simpli pentru detectarea căderilor, dar lipsa unei infrastructuri avansate limitează funcționalitățile disponibile [10].

Un alt exemplu notabil este integrarea algoritmilor de inteligență artificială (IA) în aplicațiile de sănătate. Algoritmii de IA sunt utilizati pentru a analiza tiparele de date colectate de la senzori

și pentru a genera alerte predictive. Spre exemplu, sistemele de monitorizare bazate pe IA pot identifica riscuri iminente de cădere sau modificări ale parametrilor vitali, permitând intervenții preventive [6].

Arhitecturile software moderne sunt concepute pentru a fi scalabile și flexibile, permitând integrarea unor noi funcționalități și tehnologii pe măsură ce acestea devin disponibile. Aceasta include, de asemenea, asigurarea confidențialității și securității datelor, utilizând criptarea avansată și autentificarea multi-factor.

Relevanța pentru aplicația propusă

Aplicația propusă în această lucrare își propune să integreze funcționalitățile oferite de senzorii mobili și arhitecturile software moderne pentru a oferi o soluție completă. Aceasta va utiliza:

- Accelerometru pentru detectarea automată a căderilor și alertarea unui dispecerat medical.
- Arhitecturi bazate pe cloud pentru stocarea securizată a datelor și gestionarea notificărilor automate.

Prin combinarea acestor tehnologii, aplicația va contribui la creșterea siguranței și la îmbunătățirea calității vietii utilizatorilor, oferind o soluție adaptată nevoilor persoanelor cu mobilitate redusă.

1.3 Cerințele și specificațiile aplicației

Pentru dezvoltarea unei aplicații eficiente care să sprijine persoanele cu mobilitate redusă, este esențial să fie definite clar cerințele funcționale și non-funcționale ale acesteia. În această secțiune, sunt detaliate caracteristicile principale ale aplicației, cu accent pe detectarea automată a incidentelor și alertarea rapidă a dispeceratului medical.

1.3.1 Cerințe funcționale

Cerințele funcționale descriu comportamentele specifice ale aplicației și modul în care aceasta interacționează cu utilizatorii și cu alte sisteme externe. Acestea sunt esențiale pentru a asigura funcționarea optimă a aplicației propuse.

Detectarea automată a căderilor

Una dintre funcționalitățile principale ale aplicației este detectarea automată a căderilor. Aceasta presupune utilizarea senzorilor de accelerometru și giroscop integrati în dispozitivele mobile pentru a analiza mișcările bruște ale utilizatorului. În cazul identificării unui tipar care sugerează o cădere, aplicația declanșează automat un proces de verificare prin intermediul unei notificări interactive. Dacă utilizatorul nu răspunde în intervalul de timp prestabilit, se generează o alertă automată.

Această funcționalitate este inspirată de tehnologii existente, precum cele utilizate în Apple Watch [12], dar este extinsă prin integrarea unei conexiuni directe cu dispeceratul medical. Algoritmii utilizati pentru detectarea căderilor sunt optimizați pentru a reduce ratele fals-pozițive și pentru a asigura o intervenție rapidă în situațiile critice.

Alertarea în timp real a dispeceratului

O altă cerință funcțională importantă este alertarea în timp real a dispeceratului medical. Această funcționalitate presupune transmiterea automată a unei notificări către un dispecerat centralizat atunci când este detectată o urgență. Notificarea include detalii precum locația utilizatorului, istoricul evenimentului și starea curentă a acestuia.

Aplicația utilizează tehnologia GPS pentru a transmite locația exactă a utilizatorului și asigură o comunicare bidirectională între dispecerat și utilizator. Această funcționalitate nu doar că reduce timpul de răspuns, dar și îmbunătățește coordonarea resurselor medicale pentru a interveni eficient.

În plus, aplicația este proiectată să integreze un sistem de prioritizare a alertelor, astfel încât urgențele critice să fie gestionate cu prioritate maximă. Funcționalitatea de alertare este compatibilă cu standardele de securitate GDPR, asigurând protecția datelor utilizatorilor.

1.3.2 Cerințe non-funcționale

Cerințele non-funcționale descriu calitățile și caracteristicile generale ale aplicației, care influențează performanța, securitatea și experiența utilizatorului. Acestea sunt esențiale pentru a asigura o utilizare eficientă și sigură a sistemului.

Timp minim de răspuns

Unul dintre obiectivele principale ale aplicației este asigurarea unui timp minim de răspuns în cazul unei urgente. Aplicația este proiectată să proceseze și să transmită alertele detectate în cîteva secunde de la identificarea unui incident. Această performanță este realizată prin utilizarea arhitecturilor software bazate pe cloud, care permit procesarea rapidă a datelor și transmiterea lor către dispeceratul medical.

Timpul de răspuns include:

- Detectarea incidentului și generarea alertei.
- Transmiterea notificării către serverul aplicației.
- Comunicarea cu dispeceratul medical, inclusiv transmiterea locației și a detaliilor relevante despre incident.

Această cerință este importantă pentru reducerea riscurilor asociate cu întârzierea intervențiilor, contribuind la creșterea sanselor de recuperare a utilizatorului.

Securitatea datelor utilizatorilor

Pentru a proteja informațiile sensibile ale utilizatorilor, aplicația respectă standardele internaționale de securitate a datelor, inclusiv Regulamentul General privind Protecția Datelor (GDPR). Datele utilizatorilor, cum ar fi locația și istoricul evenimentelor, sunt criptate atât în timpul transmisiei, cât și în timpul stocării.

Aplicația implementează următoarele măsuri pentru asigurarea securității:

- **Criptarea datelor:** Toate informațiile sunt criptate folosind algoritmi avansați (ex: AES-256) pentru a preveni accesul neautorizat.
- **Autentificare:** Utilizatorii și personalul dispeceratului sunt autentificați prin metode sigure.

- **Controlul accesului:** Datele sunt accesibile doar personalului autorizat, iar toate acțiunile sunt jurnalizate pentru a asigura trasabilitatea.
- **Back-up periodic:** Datele stocate pe serverele cloud sunt salvate periodic pentru a preveni pierderea informațiilor critice.

Aceste măsuri asigură integritatea și confidențialitatea datelor utilizatorilor, protejându-i împotriva amenințărilor cibernetice și a accesului neautorizat.

1.3.3 Obiectivele soluției propuse

Pentru a sprijini persoanele cu mobilitate redusă și a le îmbunătăți calitatea vieții, aplicația propusă are două obiective principale: crearea unui sistem sigur și accesibil, precum și simplificarea utilizării pentru utilizatorii vulnerabili. Aceste obiective reflectă atât nevoile identificate în analiza realizărilor existente, cât și cerințele specifice ale utilizatorilor.

Oferirea unui sistem sigur și accesibil

Unul dintre obiectivele fundamentale ale aplicației este asigurarea unui mediu sigur și accesibil pentru toți utilizatorii. Siguranța este garantată prin implementarea unor tehnologii avansate, precum detectarea automată a incidentelor, notificările rapide și integrarea cu dispeceratul medical. Accesibilitatea este abordată prin integrarea unui limbaj simplificat, adică interfața utilizatorului este intuitivă și ușor de utilizat, chiar și pentru persoanele cu un nivel scăzut de alfabetizare digitală.

Simplificarea utilizării pentru utilizatorii vulnerabili

Având în vedere că aplicația este destinată persoanelor cu mobilitate redusă, aceasta este proiectată astfel încât să fie ușor de utilizat, fără a necesita un nivel avansat de cunoștințe tehnice. Funcționalitățile cheie includ:

- **Acces rapid la funcții critice:** Funcționalitățile de detectare a căderilor și alertare sunt disponibile direct din ecranul principal al aplicației.
- **Integrarea notificărilor vocale:** Aplicația oferă mesaje audio pentru utilizatorii cu dificultăți de vedere sau alte dizabilități care le limitează utilizarea ecranului.

Aceste caracteristici asigură o experiență fluidă și intuitivă pentru utilizatori, permitându-le să beneficieze de toate funcționalitățile aplicației fără efort suplimentar. Prin design-ul său prietenos și tehnologia integrată, aplicația contribuie la reducerea barierelor digitale și la îmbunătățirea autonomiei utilizatorilor vulnerabili.

Capitolul 2 Proiectarea aplicației

Acest capitol explorează în detaliu procesul de proiectare al aplicației, având ca punct de plecare analiza platformei hardware, urmată de descrierea modulelor generale ale aplicației și a interacțiunilor dintre acestea, și încheind cu justificarea alegerilor tehnologice utilizate în implementare. Structura acestui capitol reflectă abordarea metodică adoptată pentru realizarea unei soluții robuste și eficiente.

Scopul principal al proiectării este de a dezvolta un sistem sigur, scalabil și accesibil, orientat către sprijinirea utilizatorilor vulnerabili. Proiectarea este ghidată de cerințele de a oferi răspunsuri în timp real, precum și de necesitatea unei experiențe utilizator optimizate. Alegerea componentelor hardware și software a fost realizată în urma unei analize atente, asigurându-se că acestea satisfac atât criteriile funcționale, cât și cele economice, fără a compromite performanța sau fiabilitatea.

2.1 Analiza platformei hardware

Platforma hardware aleasă pentru dezvoltarea aplicației este un smartphone modern, datorită combinației ideale de funcționalități avansate, accesibilitate și costuri reduse. Această alegere oferă toate resursele necesare pentru implementarea funcționalităților aplicației fără a necesita echipamente suplimentare, fiind o soluție practică și eficientă.

2.1.1 Senzori integrați

Senzorii reprezintă componente esențiale ale smartphone-urilor moderne, având rolul de a monitoriza interacțiunile utilizatorilor și de a detecta modificările din mediul înconjurător. Aceștia generează date care sunt utilizate în aplicații diverse, de la monitorizarea sănătății la aplicații de navigație. În cadrul aplicației dezvoltate, accelerometrul este senzorul principal utilizat pentru detectarea căderilor.

Ce este un senzor?

Senzorii sunt dispozitive capabile să detecteze modificări ale parametrilor fizici, cum ar fi accelerarea, lumina, temperatura sau orientarea dispozitivului, transformând aceste informații în semnale procesabile electronic. În smartphone-uri, senzorii permit monitorizarea mișcărilor, detectarea poziției și analizarea condițiilor mediului înconjurător [14].

Accelerometrul în aplicație

Accelerometrul măsoară accelerarea liniară pe cele trei axe principale ale dispozitivului: X, Y și Z. În cadrul aplicației, acesta monitorizează constant mișcările dispozitivului pentru a detecta variațiile bruste care pot semnala o cădere. Datele brute furnizate de accelerometru sunt utilizate

pentru calcularea magnitudinii accelerării, conform formulei:

$$\text{Magnitudine} = \sqrt{x^2 + y^2 + z^2} \quad (2.1)$$

Dacă magnitudinea accelerării depășește un prag prestabilit, aplicația consideră că a avut loc un impact semnificativ. Acest prag este configuriabil și poate fi ajustat în funcție de cerințele aplicației și de specificațiile utilizatorului.

Accelerometrul detectează mișările dispozitivului pe cele trei axe principale, reprezentate în figura 2.1.

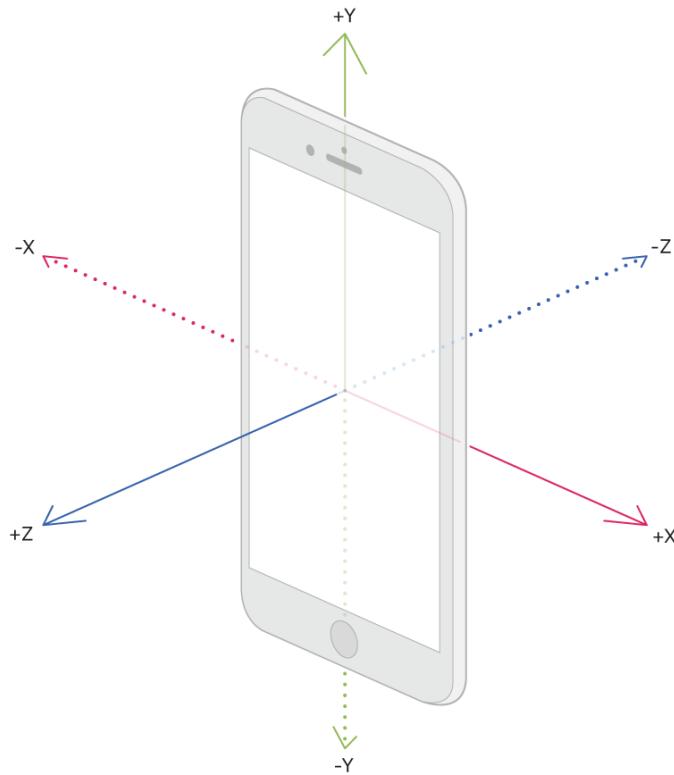


Figure 2.1: Reprezentarea axelor accelerometrului într-un smartphone modern [15].

Accelerometrul contribuie la monitorizarea continuă a dispozitivului și la detectarea căderilor prin următoarele funcționalități:

- **Detectarea căderilor:** Analizează datele brute ale accelerării pentru a identifica variațiile brusă de mișcare caracteristice impacturilor semnificative.
- **Contribuția la siguranța utilizatorului:** Permite identificarea rapidă a situațiilor critice, sprijinind astfel intervențiile în timp util.

Accelerometrul este un senzor esențial în arhitectura aplicației dezvoltate, contribuind la monitorizarea continuă și la detectarea rapidă a evenimentelor critice. Limitările sale sunt gestionate eficient prin stabilirea unui prag adecvat și prin integrarea funcționalităților de confirmare a stării utilizatorului.

2.1.2 Conectivitate avansată

Conectivitatea avansată este un element fundamental în arhitectura aplicației dezvoltate,

asigurând transmiterea datelor între aplicația mobilă, backend și dispecerat. Aceasta permite sincronizarea în timp real și transmiterea rapidă a alertelor critice, contribuind la siguranța utilizatorilor vulnerabili.

Rolul conectivității în aplicație

În cadrul aplicației, conectivitatea joacă un rol esențial în:

- **Transmiterea alertelor critice:** Aplicația mobilă trimite notificări către backend, care procesează datele și le transmite dispeceratului.
- **Sincronizarea în timp real:** Datele despre locația utilizatorului și evenimentele detectate sunt actualizate continuu în backend.
- **Accesibilitatea platformei web:** Personalul dispeceratului poate accesa informațiile utilizatorilor printr-o interfață web securizată.

Tipuri de conectivitate utilizate

Aplicația utilizează două tipuri principale de conectivitate pentru a asigura funcționarea optimă în diferite scenarii:

- **Wi-Fi:** Conexiunile Wi-Fi sunt utilizate pentru transferul rapid al datelor în medii cu rețele stabilă. Această tehnologie este ideală pentru reducerea consumului de date mobile și pentru gestionarea sincronizării în timp real. [16]
- **Date mobile (4G/5G):** Rețelele mobile permit funcționarea aplicației în medii fără acces Wi-Fi, asigurând transmiterea alertelor în timp real. Tehnologia 5G oferă viteze mai mari de transfer și latențe reduse, contribuind la răspunsuri rapide în situații de urgență. [17]

Integrarea conectivității în arhitectura aplicației

Conectivitatea avansată este integrată în arhitectura aplicației pentru a facilita:

- **Transmiterea alertelor detectate:** În momentul detectării unei căderi, aplicația trimite o notificare către backend, care conține informații despre locația utilizatorului.
- **Accesul și gestionarea datelor:** Datele utilizatorilor sunt stocate în backend și sunt accesibile prin platforma web pentru monitorizare și intervenție.
- **Răspunsul rapid din partea dispeceratului:** Notificările critice sunt procesate în timp real și afișate pe tabloul de bord al platformei web, asigurând o intervenție promptă.

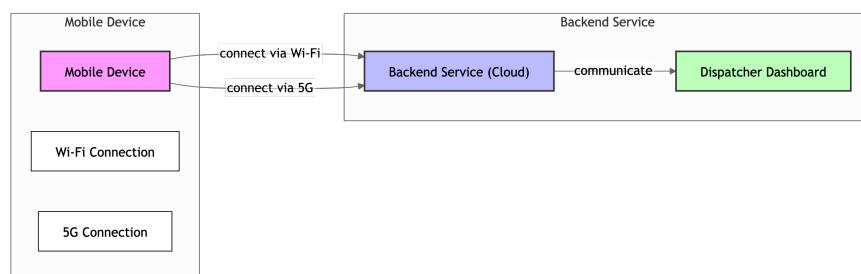


Figure 2.2: Fluxul de conectivitate între aplicație, backend și dispecerat.

Conecțivitatea avansată joacă un rol important în arhitectura aplicației, asigurând transmiterea rapidă și sigură a datelor între componentele sale. Integrarea tehnologiilor Wi-Fi și 5G contribuie la crearea unui sistem eficient și accesibil, care răspunde cerințelor utilizatorilor vulnerabili.

2.1.3 Putere de procesare

Puterea de procesare a smartphone-urilor moderne joacă un rol esențial în arhitectura aplicației dezvoltate. Aceasta permite analiza datelor senzorilor în timp real, rularea algoritmilor avansați și gestionarea eficientă a fluxurilor de date. Performanța hardware transformă smartphone-urile în dispozitive versatile, capabile să gestioneze sarcini complexe și să ofere funcționalități avansate utilizatorilor [18].

Arhitectura procesorului și multitasking-ul eficient

Procesoarele smartphone-urilor moderne sunt proiectate să ofere performanță optimă printr-o combinație echilibrată între nuclee de înaltă performanță și nuclee eficiente energetic. Această arhitectură permite dispozitivelor să gestioneze simultan sarcini complexe și procese de fundal, fără a compromite autonomia bateriei sau fluiditatea aplicațiilor.

- **Nuclee de înaltă performanță:** Specializate în rularea sarcinilor complexe, cum ar fi analiza datelor brute de la senzori și execuția algoritmilor avansați. Sunt responsabile pentru procesarea în timp real a datelor brute de la accelerometru, asigurând detectarea rapidă a căderilor. [19].
- **Nuclee eficiente energetic:** Proiectate pentru gestionarea proceselor de fundal, cum ar fi monitorizarea constantă a senzorilor sau sincronizarea aplicației cu backend-ul. Aceste nuclee contribuie la prelungirea autonomiei bateriei printr-un consum energetic redus [18].

Această separare a sarcinilor între nuclee este ilustrată în figura 2.3, care prezintă arhitectura procesorului multi-core. Nucleele de înaltă performanță sunt responsabile pentru analiza intensivă a datelor și execuția rapidă a proceselor critice, în timp ce nucleele eficiente energetic gestionează procesele secundare.

Rolul DSP-urilor în procesare

Un alt element esențial al arhitecturii procesorului este integrarea unităților DSP (Digital Signal Processor). Aceste componente sunt specializate pentru:

- DSP-ul preia datele brute de la senzori, cum ar fi accelerometrul, și le prelucrează rapid pentru a reduce sarcina procesorului principal.
- Optimizează consumul energetic, permitând rularea aplicației pe dure lungi fără compromisarea performanței.
- Sprijină algoritmii aplicației prin procesarea rapidă a semnalelor critice și transmiterea acestora către backend pentru stocare și notificări.

DSP-ul este poziționat ca un punct central în fluxul de procesare a datelor, preluând și optimizând sarcinile critice înainte ca acestea să fie transmise către backend. Această abordare combinată asigură o utilizare eficientă a resurselor hardware și îmbunătățește experiența utilizatorului.

Integrarea GPU-urilor și AI pentru performanță sporită

Pe lângă CPU și DSP, unitățile GPU contribuie la accelerarea procesării sarcinilor care implică învățarea automată sau analiza datelor complexe. Integrarea AI în procesoarele moderne, prin unități de procesare neurală (NPU), permite:

- Rularea locală a algoritmilor de învățare automată.
- Reducerea latenței și a dependenței de conexiunea la internet.
- Optimizarea funcționalităților personalizate, cum ar fi notificările bazate pe analiza vocală [20].

Fluxul procesării în aplicația propusă

Arhitectura procesorului multi-core este integrată direct în funcționalitatea aplicației dezvoltate:

1. **Detectarea căderilor:** Datele brute de la accelerometru sunt procesate de DSP pentru a identifica variațiile critice ale accelerării.
2. **Sincronizarea cu backend-ul:** Datele procesate sunt transmise către backend pentru stocare și notificări.
3. **Gestionarea resurselor:** GPU și NPU asigură optimizarea sarcinilor vizuale și AI, reducând sarcina pe nucleele principale.

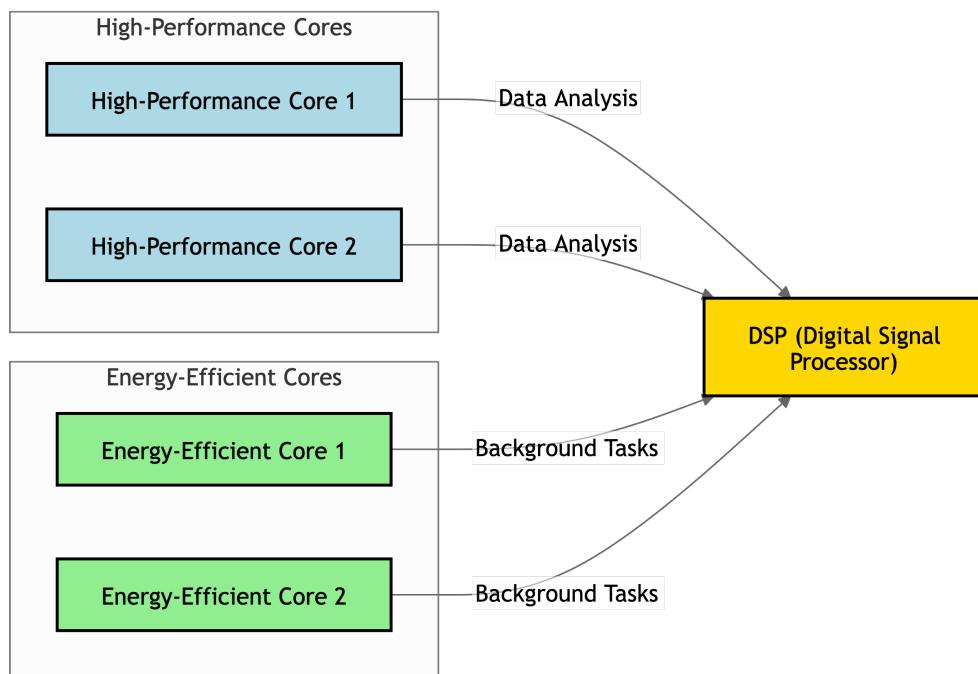


Figure 2.3: Arhitectura procesorului multi-core și fluxul procesării sarcinilor în aplicația mobilă.

Figura 2.3 ilustrează distribuția sarcinilor între componente hardware ale procesorului, demonstrând modul în care nucleele și DSP-urile colaborează pentru a optimiza fluxul de date.

Limitările hardware și optimizarea aplicației

Performanța aplicației poate varia în funcție de specificațiile hardware ale dispozitivului utilizatorului. Dispozitivele mai vechi, cu procesoare mai puțin performante, pot întâmpina dificultăți în rularea simultană a funcționalităților complexe. Optimizarea algoritmilor aplicatiei este crucială pentru a asigura compatibilitatea cu o gamă largă de dispozitive, menținând totodată eficiența energetică [21].

Puterea de procesare a smartphone-urilor moderne joacă un rol determinant în succesul aplicației dezvoltate. Arhitectura multi-core, DSP-urile și integrarea AI susțin funcționalitatea aplicației, oferind o soluție eficientă, sigură și adaptabilă nevoilor utilizatorilor vulnerabili.

2.2 Modulele generale ale aplicației și interacțiunile

Sistemul propus este structurat modular pentru a asigura o funcționare eficientă și o integrare optimă între componente principale. Aplicația este compusă din trei module principale: aplicația mobilă, backend-ul și platforma web. Fiecare dintre aceste componente îndeplinește roluri specifice, fiind concepută pentru a răspunde cerintelor funcționale și non-funcționale identificate în faza de proiectare.

Interacțiunea dintre module este facilitată de API-uri RESTful și notificări în timp real prin Socket.IO. API-urile RESTful permit transmiterea de date între aplicație, backend și platforma web, fiind utilizate pentru autentificare, raportarea incidentelor și gestionarea profilurilor pacienților. Notificările în timp real asigură o reacție rapidă la evenimentele critice, cum ar fi o cădere detectată, prin alertarea imediată a dispeceratului.

Această organizare modulară asigură:

- **Scalabilitate**, prin posibilitatea de extindere a funcționalităților.
- **Flexibilitate**, datorită separării clare a rolurilor fiecărui modul.
- **Securitate sporită**, prin utilizarea mecanismelor de autentificare și autorizare pentru protejarea datelor utilizatorilor.

În acest subcapitol, vom analiza structura fiecărui modul, fluxurile de date și modul în care acestea colaborează pentru a oferi o soluție completă și eficientă.

2.2.1 Descrierea modulelor principale

Sistemul este compus din trei module principale, fiecare proiectat pentru a îndeplini un set de funcționalități specifice:

1. **Aplicația mobilă**: Detectarea căderilor și comunicarea cu backend-ul.
2. **Backend-ul**: Gestionarea datelor și coordonarea fluxului de informații.
3. **Platforma web**: Vizualizarea alertelor și accesul la informațiile pacienților.

Fiecare modul are un rol clar definit, iar interacțiunile dintre acestea sunt fundamentale pentru buna funcționare a sistemului. În continuare, vom detalia funcționalitățile fiecărui modul.

1. Aplicația mobilă

Aplicația mobilă reprezintă componenta principală prin care utilizatorii interacționează direct cu sistemul. concepută pentru a fi intuitivă și accesibilă, aceasta oferă funcționalități esențiale, cum ar fi detectarea automată a căderilor, notificarea utilizatorului și transmiterea datelor către backend.

Interfața aplicației este simplă, fiind destinată utilizatorilor cu nevoi diverse, inclusiv persoanelor vulnerabile. Designul interfeței permite utilizatorilor să navegheze rapid între funcțiile principale, cum ar fi logarea sau înregistrarea. Ecranul principal al aplicației, ilustrat mai jos, prezintă aceste opțiuni într-un mod clar și accesibil, contribuind la o experiență prietenoasă pentru utilizator.

Funcționalități principale:

1. Detectarea căderilor:

- Senzorul dispozitivului, accelerometrul, este utilizat pentru a monitoriza mișările utilizatorului în timp real.
- Algoritmii din aplicație analizează datele colectate pentru a identifica mișările bruste sau impacturile care pot indica o cădere.

2. Interacțiunea cu utilizatorul:

- Aplicația trimite notificări locale (vizuale și audio) pentru a verifica starea utilizatorului imediat după un eveniment detectat.
 - În caz de orice tip de impact, sistemul escaladează notificarea către backend pentru a alerta dispeceratul.
3. **Transmiterea alertelor:** Informații precum locația utilizatorului, detaliile incidentului și răspunsurile utilizatorului sunt transmise backend-ului, asigurând un timp de reacție minim.
4. **Accesibilitate:** Aplicația este proiectată să fie simplu de utilizat chiar și de persoane cu un nivel scăzut de alfabetizare digitală, având o interfață clară și intuitivă.



Figure 2.4: Interfața principală a aplicației mobile, unde utilizatorii pot alege opțiunea de logare sau înregistrare.

Dezvoltată folosind limbajul **Swift**, aplicația profită de capabilitățile native ale sistemelor iOS, ceea ce permite integrarea optimă cu senzorii dispozitivului și gestionarea eficientă a resurselor. Prin aceste funcționalități, aplicația mobilă asigură monitorizarea continuă și facilitează intervențiile rapide, fiind un punct esențial de interacțiune în cadrul sistemului integrat.

2. Backend

Backend-ul reprezintă componenta centrală a sistemului, fiind responsabil de procesarea datelor, gestionarea fluxului de informații între aplicația mobilă și platforma web, și coordonarea interacțiunilor cu baza de date. Arhitectura sa modulară permite integrarea eficientă a tuturor funcționalităților esențiale, asigurând securitatea și scalabilitatea necesare pentru o utilizare optimă a sistemului.

Funcționalități principale:

1. Colectarea și procesarea datelor:

- Backend-ul primește informații esențiale de la aplicația mobilă, precum locația utilizatorului, detalii despre incidente și confirmarea stării acestuia.
- Datele sunt procesate în timp real și analizate pentru a genera notificări relevante sau pentru a actualiza istoricul evenimentelor.

2. Stocarea datelor în baza de date:

- Informațiile procesate sunt stocate într-o bază de date securizată, gestionată printr-un sistem MySQL.
- Structura bazei de date permite organizarea eficientă a informațiilor despre utilizatori, incidente și locații, garantând acces rapid și protecția datelor.

3. Gestionarea notificărilor către platforma web:

- După procesarea datelor, backend-ul transmite notificări către platforma web utilizând protocole de comunicare în timp real (de exemplu, *Socket.IO*).
- Notificările includ detalii despre utilizator, istoricul incidentelor detectate și locația acestuia în momentul incidentului.

4. Sincronizarea datelor între module:

- Backend-ul asigură sincronizarea continuă a datelor între aplicația mobilă și platforma web.
- Personalul medical are acces în timp real la informațiile actualizate.

5. Suport pentru scalabilitate și securitate:

- Sistemul backend este proiectat să gestioneze un volum mare de cereri simultane, asigurând răspunsuri rapide și utilizarea optimă a resurselor serverului.
- Sunt implementate măsuri avansate de securitate, precum criptarea datelor și autentificarea utilizatorilor prin token-uri JWT.

Fluxul de date în backend

Fluxul de date între backend și celelalte componente este ilustrat în figura de mai jos:

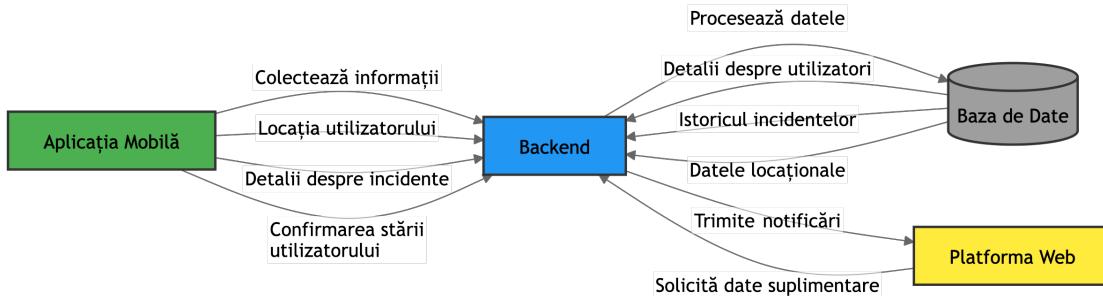


Figure 2.5: Fluxul de date între aplicația mobilă, backend și platforma web

Beneficii oferite de backend:

- Performanță optimă:** Datorită arhitecturii scalabile, backend-ul poate gestiona cereri multiple fără întreruperi.
- Securitate ridicată:** Criptarea datelor și măsurile stricte de autentificare asigură protecția informațiilor sensibile ale utilizatorilor.
- Flexibilitate:** Backend-ul permite adăugarea rapidă de noi funcționalități sau integrarea cu alte sisteme fără afectarea stabilității.

3. Platforma Web

Platforma web este un modul esențial al sistemului, oferind personalului dispeceratului instrumentele necesare pentru a monitoriza și gestiona incidentele raportate în timp real. Aceasta este proiectată pentru a integra eficient funcționalități critice, asigurând totodată securitatea și accesibilitatea informațiilor.

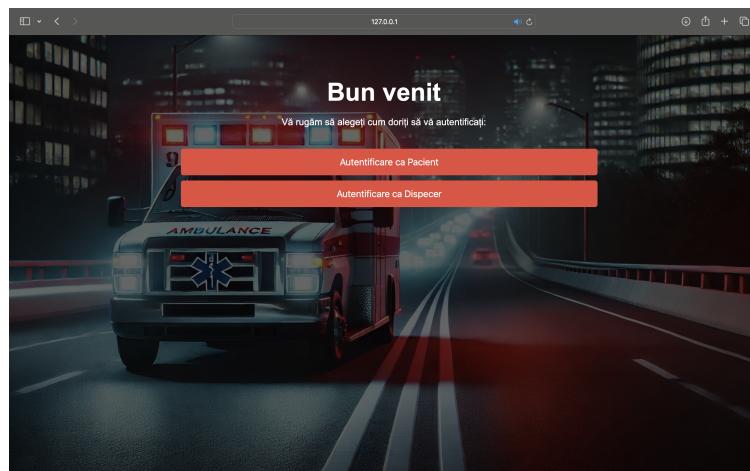


Figure 2.6: Pagina principală a platformei web, care permite autentificarea utilizatorilor.

Functionalități principale

- Afișarea alertelor în timp real:** Platforma afișează instantaneu alertele critice trimise de aplicația mobilă. Fiecare alertă conține informații precum locația utilizatorului, timpul generării și starea detectată. Aceste alerte sunt afișate temporar, fără a fi stocate, și permit dispecerilor să ia măsuri rapide.

Exemplu de utilizare: Dispecerul vizualizează o alertă critică generată de o cădere detectată și folosește informațiile furnizate pentru a trimite rapid o echipă de intervenție.

- Vizualizarea utilizatorilor activi:** Platforma permite dispecerilor să acceseze rapid lista utilizatorilor activi, împreună cu informațiile lor relevante, precum locația curentă, numele și detalii de contact. Această funcționalitate este crucială pentru identificarea rapidă a cazurilor critice.

Exemplu de utilizare: Dispecerul verifică locația unui utilizator activ pentru a coordona mai bine resursele medicale disponibile.

- Tablou de bord centralizat:** Interfața principală afișează toate alertele active într-un format simplu, evidențiind informații esențiale precum severitatea incidentului, locația și timpul generării.

Exemplu de utilizare: Dispecerul accesează tabloul de bord pentru a vizualiza rapid toate alertele active și a identifica cele mai urgente cazuri.

- Sincronizarea datelor în timp real:** Platforma este sincronizată continuu cu backend-ul și aplicația mobilă, ceea ce garantează că orice modificare sau alertă este afișată instantaneu.

Exemplu de utilizare: O schimbare în locația utilizatorului este actualizată în timp real pe platformă, permitând dispecerului să coordoneze eficient echipele de intervenție.

- Autentificare securizată:** Accesul la platformă este permis doar utilizatorilor autenticați, utilizând metode sigure, cum ar fi token-uri *JWT*. Acest sistem asigură protecția informațiilor sensibile împotriva accesului neautorizat.

Exemplu de utilizare: Un dispecer se autentifică folosind credențiale securizate și obține acces la funcționalitățile platformei, inclusiv la vizualizarea alertelor active.

Măsuri de securitate

Pentru a proteja datele utilizatorilor, platforma web implementează următoarele măsuri de securitate:

- Criptare avansată:** Parolele utilizatorilor sunt stocate folosind algoritmi de criptare precum *bcrypt*.
- Conexiuni securizate:** Toate comunicațiile sunt protejate prin protocolul *HTTPS*.
- Autorizare bazată pe roluri:** Accesul la diferite funcționalități este restricționat în funcție de rolul utilizatorului (ex. dispecer sau pacient).

Exemplu de securitate: Sistemul blochează automat accesul unui utilizator după mai multe încercări nereușite de autentificare, prevenind atacurile brute-force.

Beneficii oferite de platforma web

- **Acces rapid la date:** Personalul dispeceratului poate accesa instantaneu informații relevante pentru a lua decizii rapide și eficiente.
- **Interfață prietenoasă:** Designul platformei facilitează utilizarea chiar și în condiții de stres, oferind un flux de lucru clar și bine organizat.
- **Securitate ridicată:** Platforma protejează datele sensibile ale utilizatorilor prin implementarea celor mai bune practici de securitate informatică.

Exemplu de beneficiu: Platforma este utilizată zilnic de personalul medical pentru a coordona intervenții rapide și eficiente, reducând timpuri de reacție în situații critice.

2.2.2 Fluxurile de date planificate

Fluxurile de date din cadrul sistemului sunt esențiale pentru asigurarea unei funcționări fluide și eficiente între modulele aplicației. Sistemul implică patru componente principale: senzori (accelerometru), aplicația mobilă, backend-ul și platforma web.

Diagramă explicativă

Diagrama prezintă fluxul de date dintre modulele aplicației. De la detectarea datelor brute de către Senzorul Accelerometru, acestea sunt prelucrate de Aplicația Mobilă, validate și stocate în Backend, și transmise către Platforma Web prin WebSockets. Dispecerat vizualizează alertele în timp real și coordonează răspunsul în funcție de severitatea incidentului. Componentele sunt evidențiate prin culori distincte pentru o mai bună înțelegere a interacțiunilor.

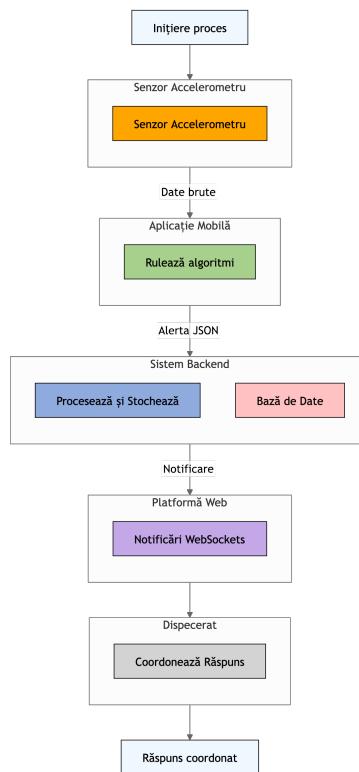


Figure 2.7: Fluxul general al datelor între senzori, aplicația mobilă, backend și platforma web.

Rolul fiecărui modul

- Aplicația mobilă: Procesează datele de la senzori și generează alerte. Este responsabilă de transmiterea datelor către backend.
- Backend-ul: Centralizează procesarea și stocarea datelor, asigurând integrarea dintre aplicația mobilă și platforma web.
- Platforma web: Afisează datele procesate și oferă unele pentru gestionarea alertelor de către dispeceri.

Descrierea fluxului general

Fluxul general al datelor poate fi descris astfel:

1. Senzorul accelerometru detectează o potențială cădere și trimit date brute către aplicația mobilă.
2. Aplicația mobilă procesează datele brute, aplică algoritmi de analiză și generează o alertă dacă este identificată o situație critică. Această alertă include locația utilizatorului (coordinate GPS), timpul evenimentului și alte date relevante.
3. Backend-ul primește datele alertei de la aplicația mobilă printr-un endpoint REST, le validează și le stochează în baza de date. De asemenea, backend-ul transmite alerta către platforma web utilizând WebSockets pentru notificări în timp real.
4. Platforma web afisează alerta în timp real pe tabloul de bord centralizat, permitând dispecerilor să ia măsuri imediate.

2.3 Proiectarea software

Proiectarea software reprezintă etapa esențială în dezvoltarea aplicației, asigurând interacțiunea eficientă dintre modulele principale și implementarea funcționalităților critice. Arhitectura aplicației a fost concepută pentru a integra senzori, algoritmi avansati și mecanisme de transmisie a datelor într-un mod scalabil și robust.

Această secțiune detaliază structura generală a aplicației, interacțiunile dintre module (Aplicația mobilă, Backend și Platforma web), organizarea bazei de date și tehnologiile utilizate pentru a îndeplini cerințele funcționale și non-funcționale. Sistemul este proiectat astfel încât să minimizeze timpii de răspuns, să asigure securitatea datelor și să fie accesibil pentru utilizatori vulnerabili.

2.3.1 Arhitectura generală a aplicației

Aplicația dezvoltată utilizează o arhitectură modulară, formată din trei componente principale interconectate: **aplicația mobilă**, **backend-ul** și **platforma web**. Aceste module sunt completate de o **bază de date MySQL** pentru gestionarea datelor critice. Fiecare componentă are un rol bine definit, contribuind la funcționarea integrată a sistemului:

- **Aplicația mobilă (Swift):**
 - Detectează căderile utilizatorilor folosind senzorii dispozitivului (accelerometru) și algoritmi de procesare locală a datelor brute.

- Trimite alerte detaliate către backend sub formă de obiecte JSON prin intermediul unui **REST API** securizat.
 - Asigură accesibilitatea pentru utilizatori printr-o interfață intuitivă.
- **Backend-ul (Flask):**
 - Gestionează colectarea și procesarea datelor transmise de aplicația mobilă.
 - Facilitează sincronizarea în timp real a alertelor și stocarea lor în baza de date MySQL.
 - Trimite notificări către platforma web utilizând **WebSockets**, oferind actualizări instantanee pentru dispeceri.
 - **Platforma web (React):**
 - Permite personalului medical să monitorizeze în timp real alertele generate.
 - Asigură gestionarea eficientă a situațiilor critice printr-un tablou de bord centralizat.
 - Include autentificare securizată bazată pe token-uri pentru protecția datelor sensibile.

Modul între modulele colaborează

Colaborarea dintre module se bazează pe tehnologii moderne și protocoale de comunicare eficiente, asigurând transferul rapid și sigur al datelor:

- **Aplicația mobilă → Backend:**
 - Datele colectate de la accelerometru sunt procesate și transmise către backend printr-un endpoint REST dedicat.
 - Alerta include informații precum:
 - * Coordonatele GPS ale utilizatorului.
 - * Timpul evenimentului.
 - * Detalii suplimentare despre starea utilizatorului.
- **Backend → Platformă web:**
 - Backend-ul utilizează WebSockets pentru a transmite în timp real alertele către platformă web.
 - Personalul medical vizualizează detaliile critice pe tabloul de bord, incluzând locația utilizatorului și severitatea incidentului.

Scheme logice

Arhitectura generală a aplicației este reprezentată schematic în figura de mai jos, evidențiind fluxul de date dintre module:

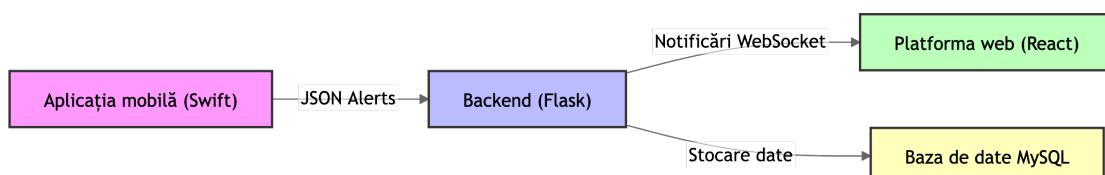


Figure 2.8: Arhitectura generală a aplicației, incluzând modulele mobile, backend și web, precum și baza de date.

Această diagramă subliniază fluxurile principale de informații, de la detectarea unui eveniment critic și până la notificarea în timp real a dispecerilor. Designul modular asigură scalabilitatea și flexibilitatea necesare pentru extinderea ulterioară a funcționalităților.

2.3.2 Diagrama ER a bazei de date

Relațiile între tabele

Baza de date utilizată de backend este proiectată pentru a susține gestionarea utilizatorilor, alertelor și logurilor, asigurând o structură logică și scalabilă.

Ce sunt diagramele ER?

Diagramele Entitate-Relatie (ER) sunt instrumente grafice utilizate pentru a modela structura logică a unei baze de date. Ele descriu entitățile principale (tabelele), atributele acestora (coloanele) și relațiile dintre ele. Acest tip de diagramă oferă o vedere de ansamblu asupra bazei de date și ajută la înțelegerea modului în care datele sunt interconectate.

Structura bazei de date include următoarele tabele principale:

- **Tabelul alembic_version:** “version_num” (VARCHAR(32)): Versiunea bazei de date, utilizată pentru a urmări modificările aduse structurii.
- **Tabelul patient:** Conține informații detaliate despre utilizatori:
 - Identificatori unici: *id* (cheie primară).
 - Date personale: *username, email, prenume, nume*.
 - Informații medicale: *boala, medicamente*.
 - Ultimul incident înregistrat: *ultimul_incident*.
 - Adresa completă: *tara, judet, oras, strada, numar, cod_postal*.
 - Locație: *latitude, longitude*.
 - Conectivitate: *este_logat, telefon, data_nasterii*.
- **Tabelul accident:** Stochează detalii despre incidente:
 - Identificator unic: *id* (cheie primară).
 - Locație: *latitude, longitude*.
 - Detalii despre incident: *details*.
 - Momentul înregistrării: *time*.
- **Tabelul dispatcher:** Gestioneză informațiile despre personalul dispeceratului:
 - Identificator unic: *id* (cheie primară).
 - Credențiale: *username, password*.

Conexiuni între tabele

Conexiunile dintre tabele sunt proiectate pentru a asigura integritatea datelor și pentru a permite interogări rapide și eficiente.

Ce înseamnă relația 1:N?

O relație de tip 1:N indică faptul că pentru fiecare entitate dintr-un tabel (partea „1”), pot exista mai multe entități asociate în alt tabel (partea „N”). De exemplu, un pacient (partea „1”) poate genera mai multe incidente (partea „N”).

- **patient → accident:**

- Relația este de tip 1:N, ceea ce înseamnă că un pacient poate genera mai multe incidente.
- Cheia externă din tabelul *accident* (*patient_id*) face referință la cheia primară din tabelul *patient*.

- **accident → dispatcher (optional):**

- Relația este de tip 1:N, asociajnd incidentele cu personalul responsabil pentru gestionarea lor.
- Cheia externă din tabelul *accident* (*dispatcher_id*) face referință la cheia primară din tabelul *dispatcher*.

Diagrama vizuală

Pentru a ilustra aceste relații, următoarea diagramă ER prezintă conexiunile dintre tabelele bazei de date:

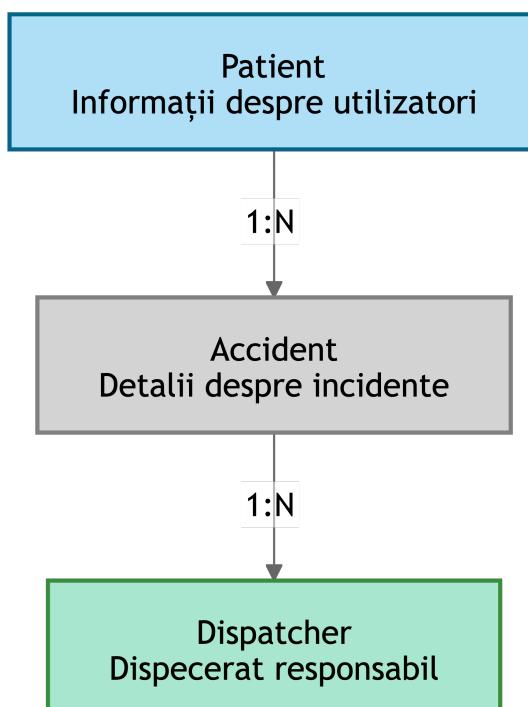


Figure 2.9: Arhitectura generală a aplicației, incluzând modulele mobile, backend și web, precum și baza de date.

Această structură permite gestionarea eficientă a informațiilor și extinderea ulterioară a bazei de date prin adăugarea altor tabele sau relații, dacă este necesar. Designul bazei de date asigură integritatea referențială, evitând inconsistențele în date.

Diagramele ER contribuie la o înțelegere clară a relațiilor dintre entități și suportă interogări eficiente pentru funcționalitățile aplicației.

2.3.3 Tehnologiile utilizate

Tehnologiile utilizate în cadrul acestui proiect au fost selectate cu scopul de a crea un sistem robust, performant și scalabil, care să răspundă cerințelor funcționale și non-funcționale definite. Alegerea acestor tehnologii reflectă un echilibru între inovație, compatibilitate și eficiență, permitând dezvoltarea și integrarea optimă a modulelor aplicației: mobilă, backend și platformă web. În cele ce urmează, vom detalia limbajele, framework-urile și instrumentele utilizate pentru fiecare componentă a sistemului.

2.3.3.1 Aplicația mobilă

Limbaj de programare: Swift

Swift este un limbaj de programare puternic și intuitiv, dezvoltat de Apple pentru a simplifica și eficientiza procesul de creare a aplicațiilor pentru platformele sale, precum iOS, iPadOS, macOS, tvOS și watchOS. Lansat oficial în 2014, Swift a fost conceput ca o alternativă mai rapidă și mai sigură la Objective-C, integrând funcționalități moderne pentru a răspunde nevoilor actuale ale dezvoltatorilor.

Caracteristici principale

- Sintaxă modernă și expresivă:** Swift oferă o sintaxă clară și concisă, care face codul mai ușor de scris și citit.
- Siguranță și gestionare automată a memoriei:** Variabilele trebuie să fie inițializate înainte de utilizare, iar operațiile critice sunt verificate pentru a preveni erori comune.
- Performanță ridicată:** Swift este optimizat pentru a funcționa rapid pe platformele Apple, permitând implementarea de aplicații complexe.
- Interoperabilitate:** Permite utilizarea codului Objective-C existent, facilitând integrarea cu proiectele mai vechi.

Framework-uri utilizate

CoreMotion este un framework dezvoltat de Apple, utilizat pentru a accesa datele senzorilor de mișcare și mediu integrate în dispozitivele iOS, iPadOS și watchOS. Acesta oferă informații detaliate despre mișcarea dispozitivului, inclusiv accelerometrul, giroscopul, magnetometrul și altimetrul. Framework-ul permite dezvoltatorilor să implementeze funcționalități avansate precum detectarea căderilor, urmărirea activității fizice, analiza orientării și măsurarea deplasării. Prin utilizarea claselor precum ‘CMMotionManager’, dezvoltatorii pot activa și configura actualizările senzorilor în timp real, integrând aceste date în aplicații pentru a îmbunătăți experiența utilizatorului.

- **CMMotionManager:** Clasa principală pentru gestionarea datelor de mișcare. Permite activarea și configurarea accelerometrului și giroscopului pentru a detecta mișcări semnificative. Este utilizată pentru a iniția actualizările senzorilor și a configura intervalele de citire a datelor.
- **CMAccelerometerData:** Oferă date brute despre acceleratia dispozitivului pe axe X, Y și Z. Aceste informații sunt folosite pentru calcularea magnitudinii acceleratiei totale, un indicator esențial pentru detectarea căderilor sau mișcărilor bruște.

UIKit este framework-ul fundamental responsabil pentru construirea și gestionarea interfeței utilizatorului în aplicațiile iOS. Acesta include o gamă variată de componente predefinite care facilitează dezvoltarea interfețelor vizuale atractive și funcționale. Framework-ul oferă suport pentru gesturi tactile, animații fluide, tranziții între ecrane și integrare perfectă cu hardware-ul Apple, fiind optimizat pentru performanță ridicată.

În combinație cu CoreMotion, UIKit permite nu doar afișarea datelor senzorilor într-un mod intuitiv, ci și interacțiunea eficientă a utilizatorilor cu funcționalitățile aplicației. Componentele principale utilizate includ:

- **UIView:** Componenta de bază pentru toate elementele grafice. Fiecare ecran al aplicației este construit prin combinarea diferitelor instanțe de `UIView`.
- **UILabel:** Utilizată pentru afișarea textului informativ, cum ar fi notificările sau alertele generate de aplicație.
- **UIButton:** Permite utilizatorilor să inițieze acțiuni, cum ar fi trimitera notificărilor către backend sau confirmarea alertei.
- **UITableView:** Utilizată pentru afișarea datelor structurate, cum ar fi o listă de incidente detectate sau istoricul alertelor.
- **NavigationController:** Facilitează navigarea între diferențele ecrane ale aplicației, oferind o experiență coerentă și intuitivă pentru utilizatori.

Astfel, CoreMotion și UIKit lucrează împreună pentru a oferi o experiență completă utilizatorilor aplicației. CoreMotion se ocupă de procesarea datelor senzorilor pentru a detecta mișările, în timp ce UIKit prezintă aceste informații într-un mod accesibil și ușor de înțeles. Această combinație permite realizarea unei aplicații mobile complexe, dar prietenoase, adaptată nevoilor utilizatorilor și obiectivelor proiectului.

Exemplu de cod din proiect

Codul de mai jos demonstrează utilizarea framework-ului CoreMotion pentru a detecta modificările bruște de mișcare, un element esențial al funcționalității aplicației:

Justificarea alegerii Swift

În cadrul acestui proiect, Swift a fost ales pentru dezvoltarea aplicației mobile datorită avantajelor sale:

- **Integrare nativă cu platforma iOS:** Compatibilitatea perfectă cu hardware-ul și software-ul Apple asigură performanță maximă și acces la API-uri native.

```

struct ContentView: View {
    func startMotionUpdates() {
        // Initializeaza procesul de actualizare a accelerometrului
        print("startMotionUpdates called")
        // Verificam dacă accelerometrul este disponibil pe dispozitiv
        if motionManager.isAccelerometerAvailable {
            print("Accelerometer is available")
            // Setăm intervalul de actualizare la 0.1 secunde pentru răspuns rapid
            motionManager.accelerometerUpdateInterval = 0.1
            // Pornim actualizarea accelerometrului
            motionManager.startAccelerometerUpdates(to: .main) { data, error in
                if let data = data {
                    // Extragem valorile de accelerare pe cele trei axe
                    let x = data.acceleration.x
                    let y = data.acceleration.y
                    let z = data.acceleration.z
                    // Calculăm magnitudinea mișării folosind formula radicalului
                    let magnitude = sqrt(x * x + y * y + z * z)

                    // Verificăm dacă magnitudinea depășește pragul stabilit (2.0)
                    if magnitude > 2.0 {
                        print("Accelerometer Data: x=\(x), y=\(y), z=\(z), magnitude=\(magnitude)")
                        // Detectăm impactul și pornim notificarea vocală dacă utilizatorul nu este deja în mișcare
                        if !isMoving {
                            isMoving = true
                            impactDetected = true
                            speechManager.startSpeaking()
                        }
                    } else {
                        // Resetez starea dacă magnitudinea este sub prag
                        isMoving = false
                    }
                } else if let error = error {
                    // Continem eventualele erori
                    print("Accelerometer error: \(error.localizedDescription)")
                } else {
                    // Cas în care nu există date disponibile
                    print("No accelerometer data")
                }
            }
        } else {
            // Mesaj afișat dacă accelerometrul nu este disponibil
            print("Accelerometer is not available")
        }
    }
}

```

Figure 2.10: Exemplu de cod pentru funcția `startMotionUpdates()` utilizând CoreMotion în Swift.

- **Comunitate activă și suport extins:** Limbajul beneficiază de actualizări regulate și resurse vaste pentru dezvoltatori.
- **Adaptabilitate:** Sintaxa expresivă și siguranța sporită facilitează implementarea unor funcționalități critice precum detectarea căderilor.

Integrarea Swift în proiect

Prin utilizarea Swift, aplicația mobilă beneficiază de un limbaj modern, sigur și eficient. Detectarea căderilor necesită procesarea datelor senzorilor în timp real, iar Swift permite implementarea algoritmilor eficienți datorită vitezei sale de execuție. Performanța superioară și compatibilitatea nativă cu dispozitivele iOS asigură o experiență optimă pentru utilizatori și facilitează trimiterea alertelor către backend fără întârzieri.

2.3.3.2 Backend

Limbaj de programare: Python

Python este un limbaj de programare interpretat, dinamic și versatil, recunoscut pentru claritatea sintaxei sale și ușurința de învățare. A fost creat în 1991 de Guido van Rossum și de-a lungul timpului a devenit unul dintre cele mai populare limbi de programare, utilizat pe scară largă în diverse domenii precum dezvoltare web, analize de date, învățare automată, automatizări și aplicații științifice.

- **Citibilitate și claritate:** Sintaxa intuitivă reduce timpul necesar pentru întreținere și crește productivitatea echipelor de dezvoltare. Codul scris în Python este ușor de înțeles, ceea ce permite altor dezvoltatori să contribuie mai rapid la proiecte existente.
- **Ecosistem bogat de librării:** Python beneficiază de o vastă colecție de librării, cum ar fi NumPy pentru procesarea numerică, Pandas pentru analiza datelor, TensorFlow pentru învățare automată și Flask pentru dezvoltare web. Acest ecosistem robust reduce semnificativ timpul necesar pentru a adăuga funcționalități complexe.

- **Comunitate activă:** O comunitate globală de dezvoltatori contribuie constant la îmbunătățirea limbajului. Documentația bine întreținută, tutorialele și ghidurile disponibile online ajută dezvoltatorii să rezolve rapid problemele și să adopte cele mai bune practici.

Framework: Flask

Flask este un framework web minimalist pentru Python, creat pentru a facilita dezvoltarea rapidă a aplicațiilor web și API-urilor RESTful. Este construit pe baza bibliotecii Werkzeug pentru gestionarea serverului și a Jinja2 pentru generarea template-urilor. Alegerea acestui framework este justificată prin mai multe avantaje:

- **Flexibilitate:** Flask nu impune o structură rigidă a proiectului, permitând dezvoltatorilor să își organizeze codul aşa cum consideră optim. Acest aspect este important pentru prototipuri și proiecte complexe care necesită arhitecturi personalizate.
- **Integrare simplă cu baze de date:** Flask se integrează perfect cu SQLAlchemy, un ORM care permite gestionarea relațiilor între tabelele bazei de date și scrierea interogărilor într-un mod intuitiv. De asemenea, Flask oferă suport nativ pentru alte tipuri de baze de date NoSQL și sisteme de caching.
- **Suport pentru notificări în timp real:** Flask permite integrarea rapidă a WebSockets, ceea ce face posibilă comunicarea bidirectională între server și client. Acest lucru este crucial pentru aplicațiile care necesită actualizări imediate sau notificări în timp real.
- **Documentație de calitate:** Ghidurile oficiale și exemplele practice ajută dezvoltatorii să înțeleagă rapid conceptele de bază și să implementeze funcționalități avansate fără a pierde timp prețios.
- **Extensibilitate:** Flask oferă numeroase extensii care facilitează adăugarea de funcționalități precum autentificare, manipulare de formulare, caching și gestionarea sesiunilor. Aceste extensii sunt bine documentate și ușor de integrat, reducând astfel complexitatea dezvoltării.

De ce Flask?

Flask a fost ales datorită flexibilității și capacitații sale de a scalabiliza proiectele pe măsură ce cerințele evoluează. Cu Flask, dezvoltatorii pot începe cu un MVP (Minimum Viable Product) rapid și pot adăuga treptat funcționalități mai complexe, fără a fi limitați de structura framework-ului. În plus, comunitatea activă și resursele disponibile online fac din Flask o soluție excelentă pentru dezvoltarea backend-ului unui proiect modern.

Combinarea dintre Python și Flask oferă o soluție robustă, flexibilă și ușor de întreținut, ideală pentru construirea unui backend performant și scalabil. Alegerea acestor tehnologii asigură un proces de dezvoltare eficient și o bază solidă pentru extinderea ulterioară a funcționalităților aplicației.

Integrarea Flask-SocketIO

Flask-SocketIO extinde funcționalitățile standard ale framework-ului Flask pentru a permite comunicații bidirectionale în timp real între backend și clienti. Această funcționalitate este esențială pentru sistemele care necesită sincronizare instantanee a datelor, cum ar fi transmiterea alertelor către dispecerat.

- **Notificări în timp real:** Serverul transmite notificări către platforma web imediat ce un incident este detectat.
- **Evenimente personalizate:** Gestionarea evenimentelor precum:

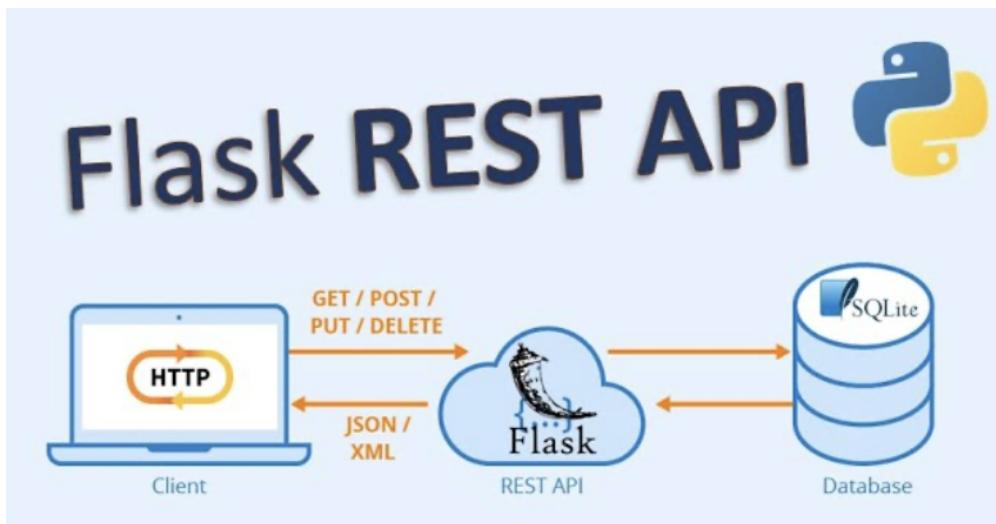


Figure 2.11: Arhitectura REST API implementată utilizând Flask.

- *connect*: Declanșat atunci când un client stabilește o conexiune cu serverul.
- *disconnect*: Declanșat când un client deconectează.
- *alert_event*: Un eveniment personalizat pentru transmiterea alertelor.

Exemplu practic

```
from flask_socketio import SocketIO, disconnect
# Inițializarea Flask-SocketIO
socketio = SocketIO(app, cors_allowed_origins="*")

@socketio.on('connect')
def handle_connect():
    print('Client connected')
    # Fără verificare de token

@socketio.on('disconnect')
def handle_disconnect():
    print('Client disconnected')
```

Figure 2.12: Exemplu de eveniment WebSocket în Flask-SocketIO.

Beneficiile utilizării Flask-SocketIO

- **Actualizări instantanee:** Platforma web primește notificări fără a reîmprospăta pagina.
- **Performanță îmbunătățită:** WebSockets sunt mai eficiente decât soluțiile clasice precum polling-ul.
- **Ușurință în utilizare:** Flask-SocketIO este bine documentat și simplu de integrat.

JWT (JSON Web Tokens)

JWT este folosit pentru autentificarea securizată a utilizatorilor. Acest mecanism permite generarea și verificarea token-urilor, asigurând că doar utilizatorii autorizați pot accesa resursele backend-ului.

- **Autentificare securizată:** Token-urile sunt generate pe baza unei chei secrete și includ informații despre utilizator, cum ar fi ID-ul acestuia. Acest lucru permite identificarea utilizatorilor într-un mod sigur.
- **Stateless:** JWT elimină necesitatea de a păstra sesiuni pe server, reducând astfel încărcarea acestuia.
- **Portabilitate:** Token-urile sunt ușor de utilizat cu diferite front-end-uri (aplicații web, mobile).

2.3.3.3 Platforma web

Platforma web reprezintă punctul de interacțiune principal pentru utilizatori, fiind dezvoltată folosind un ecosistem modern de tehnologii. Limbajul de programare utilizat este JavaScript, în combinație cu framework-ul React, pentru a construi o interfață dinamică și eficientă. HTML și CSS oferă suportul structural și vizual necesar, iar biblioteca Axios asigură transferul rapid și fiabil de date între client și backend.

Limbaj de programare: JavaScript

JavaScript este baza platformei web, fiind folosit pentru logica aplicației și interactivitatea interfetei. Alegerea limbajului a fost influențată de flexibilitatea sa și de ecosistemul extins de librării și framework-uri. Printre acestea, React a fost ales pentru capacitatea sa de componentizare, care permite dezvoltarea modulară și ușor de întreținut a aplicației.

- **Flexibilitate:** JavaScript oferă suport pentru multiple paradigmă de programare, inclusiv orientată pe obiecte și funcțională.
- **Performanță:** Este optimizat pentru execuția directă în browsere, reducând latența și oferind o experiență fluidă utilizatorilor.
- **Ecosistem vast:** JavaScript beneficiază de un ecosistem extins, incluzând librării și framework-uri precum React, Angular și Vue.

Framework-uri utilizate

În **React**, fiecare parte a interfetei este construită sub formă de componente reutilizabile, ceea ce simplifică gestionarea codului și scalarea aplicației. El a fost ales pentru dezvoltarea unei interfețe web moderne și dinamice datorită mai multor avantaje cheie:

- **Componentizare:** React permite împărțirea interfetei utilizatorului în componente mici și reutilizabile, ceea ce facilitează dezvoltarea și întreținerea codului. Fiecare componentă gestionează propria logică și propriul stil, oferind astfel flexibilitate și modularitate.
- **Performanță:** React utilizează Virtual DOM, ceea ce optimizează procesele de actualizare a interfeței în medii cu multe modificări dinamice de date. Acest lucru face ca aplicațiile dezvoltate să fie rapide și responsive.

De ce React?

- **Interfețe responsive și intuitive:** React oferă suport pentru dezvoltarea unor interfețe ușor de utilizat și adaptabile la diferite dimensiuni de ecran, ceea ce este esențial pentru o experiență optimă a utilizatorilor.
- **Comunitate extinsă:** Cu o comunitate mare de dezvoltatori, React beneficiază de un ecosistem bogat de biblioteci, unele și resurse. Suportul extins oferit de comunitate facilitează rezolvarea problemelor și adoptarea celor mai bune practici.

Beneficiile utilizării React

- **Reutilizarea codului:** Componentele create pot fi reutilizate în mai multe secțiuni ale aplicației, reducând redundanța și timpul de dezvoltare.
- **Performanță în timp real:** Integrarea React cu WebSockets sau API-urile REST asigură actualizări rapide ale datelor pe interfață.
- **Flexibilitate:** React poate fi combinat cu alte biblioteci sau framework-uri, cum ar fi Redux pentru gestionarea stării sau Axios pentru apeluri API.

React oferă o soluție robustă și modernă pentru dezvoltarea unei platforme web dinamice, responsive și ușor de întreținut. Datorită flexibilității și ecosistemului său extins, React asigură o experiență excelentă atât pentru dezvoltatori, cât și pentru utilizatori.

HTML și CSS – structura și stilul aplicației

Pentru ca React să funcționeze eficient, structura paginilor este definită folosind **HTML**, iar aspectul vizual este controlat prin **CSS**. HTML este utilizat pentru a organiza conținutul paginilor, inclusiv elemente precum titluri, paragrafe și liste. În schimb, CSS adaugă stil și personalitate aplicației, controlând culorile, fonturile, marginile și alte aspecte estetice.

Axios – conectivitatea cu backend-ul

Pentru comunicarea eficientă dintre platforma web și backend, a fost utilizată biblioteca **Axios**, care facilitează apelurile API. Aceasta permite transferul rapid de date, fiind integrată nativ în aplicație pentru a interacționa cu backend-ul dezvoltat în Flask. Axios este folosit, de exemplu, pentru a obține lista de pacienți din baza de date:

Platforma web combină tehnologiile HTML, CSS, JavaScript, React și Axios pentru a crea o aplicație modernă, performantă și ușor de utilizat. HTML și CSS oferă baza structurală și vizuală, în timp ce React și JavaScript aduc dinamism și interactivitate. Axios joacă un rol esențial în conectarea interfeței cu backend-ul, asigurând sincronizarea eficientă a datelor. Împreună, aceste tehnologii formează un ecosistem solid, adaptat cerințelor proiectului.

2.3.3.4 Baza de date

Tipul bazei de date: MySQL

MySQL este recunoscută pentru fiabilitatea și performanța sa în gestionarea datelor structurate, ceea ce o face ideală pentru proiectul propus. Este un sistem de gestionare a bazelor de date relaționale open-source, utilizat pe scară largă datorită stabilității și scalabilității sale.

- **Fiabilitate:** MySQL asigură integritatea datelor și permite gestionarea în siguranță a informațiilor critice ale aplicației.
- **Performanță:** Este optimizat pentru gestionarea interogărilor complexe și oferă viteze rapide de acces la date, chiar și pentru baze de date mari.
- **Flexibilitate:** Oferă suport pentru o gamă variată de tipuri de date și permite definirea relațiilor complexe între tabele.

De ce MySQL?

- **Relații complexe între tabele:** MySQL permite definirea și gestionarea relațiilor complexe, ceea ce este esențial pentru modelarea datelor din aplicație.
- **Performanță optimizată:** Este capabil să proceseze interogări complexe cu timpi de răspuns foarte mici, ceea ce îl face potrivit pentru aplicații cu încărcare mare.
- **Suport pentru integritatea referențială:** Funcționalitățile de gestionare a cheilor externe și a relațiilor între tabele asigură consistența datelor.

Exemplu de structură a tabelului patient

Structura tabelului patient este prezentată în Figura 2.13 și include toate informațiile relevante despre utilizatori, de la date personale la informații despre locație și conectivitate.

```
CREATE TABLE "patient" (
    id INTEGER NOT NULL,
    username VARCHAR(150) NOT NULL,
    email VARCHAR(150) NOT NULL,
    password VARCHAR(150) NOT NULL,
    prenume VARCHAR(150),
    nume VARCHAR(150),
    boala VARCHAR(500),
    medicamente VARCHAR(500),
    ultimul_incident VARCHAR(500),
    tara VARCHAR(100),
    judet VARCHAR(100),
    oras VARCHAR(100),
    strada VARCHAR(200),
    numar VARCHAR(50),
    cod_postal VARCHAR(20),
    latitude FLOAT,
    longitude FLOAT,
    este_logat BOOLEAN,
    telefon VARCHAR(20),
    data_nasterii DATE,
    PRIMARY KEY (id),
    UNIQUE (email),
    UNIQUE (username)
```

Figure 2.13: Structura tabelului patient în baza de date MySQL.

Beneficiile utilizării MySQL

- **Scalabilitate:** MySQL poate gestiona baze de date mari, fiind ideal pentru aplicații care se așteaptă să crească în timp.

- **Comunitate activă:** Cu o bază largă de utilizatori, MySQL beneficiază de suport extins și resurse de învățare.
- **Compatibilitate:** MySQL este compatibil cu majoritatea limbajelor de programare, inclusiv Python, ceea ce facilitează integrarea cu backend-ul.

MySQL oferă o soluție fiabilă și performantă pentru gestionarea datelor structurate din aplicație. Datorită funcționalităților sale avansate, MySQL contribuie semnificativ la asigurarea consistenței și accesibilității informațiilor critice.

Capitolul 3 Implementarea aplicației și rezultatele obținute

Acest capitol detaliază procesul tehnic de implementare al aplicației propuse, punând accent pe integrarea eficientă a componentelor sistemului și pe soluțiile utilizate pentru a depăși dificultățile întâmpinate. Spre deosebire de capitolele anterioare, unde s-a discutat proiectarea generală și funcționalitățile principale, aici se oferă o perspectivă detaliată asupra modului în care aplicația a fost construită.

Implementarea a fost ghidată de obiectivul de a crea un sistem robust și scalabil, care să asigure detectarea rapidă și precisă a incidentelor prin aplicația mobilă, gestionarea eficientă a datelor și notificărilor prin intermediul backend-ului și vizualizarea intuitivă a alertelor și utilizatorilor prin platforma web.

Capitolul este structurat astfel încât să ofere o imagine completă a procesului tehnic, incluzând descrierea generală a implementării, cu accent pe contribuția fiecărei componente la funcționalitatea globală, problemele întâmpinate în timpul dezvoltării și soluțiile aplicate pentru depășirea acestora, fluxul general de funcționare al aplicației, ilustrat cu capturi de ecran și exemple relevante, precum și măsurile luate pentru comunicarea între sisteme și stocarea sigură a informațiilor.

Aceste detalii oferă o perspectivă practică asupra complexității procesului de implementare, evidențiind modul în care tehnologiile alese au fost utilizate pentru a îndeplini cerințele proiectului.

Implementarea aplicației a fost realizată prin integrarea eficientă a celor trei componente principale: aplicația mobilă, backend-ul și platforma web. Fiecare dintre acestea contribuie la funcționalitatea globală a sistemului, asigurând detectarea incidentelor, gestionarea datelor și notificărilor, precum și vizualizarea intuitivă a alertelor.

3.1 Aplicația mobilă – Etape ale implementării

Implementarea aplicației mobile a fost realizată conform unui proces structurat, urmând mai multe etape distințe. Acestea au asigurat o dezvoltare organizată, eficientă și aliniată cerințelor proiectului.

3.1.1 Inițializarea proiectului

Implementarea aplicației mobile a început prin configurarea unui proiect nou în **Xcode**, folosind limbajul **Swift**. Această etapă a inclus stabilirea arhitecturii de bază, adăugarea bibliotecilor necesare și configurarea permisiunilor esențiale pentru funcționarea aplicației.

3.1.1.1 Arhitectura aplicației mobile

Pentru implementarea aplicației mobile, s-a adoptat o **arhitectură modulară**, care separă funcționalitățile esențiale în componente distințe. Această structură permite o mai bună organizare a codului, facilitează întreținerea și scalabilitatea proiectului și reduce interdependențele dintre module. Alegerea acestei arhitecturi este justificată prin necesitatea de a asigura o dezvoltare eficientă și de a permite extinderea aplicației pe viitor.

Structura proiectului Aplicația este organizată în mai multe module principale, aşa cum se observă în structura proiectului din Figura 3.1:

- **Models:**

- Include structuri și clase pentru gestionarea datelor persistente.
- Subfolderul **Persistence** centralizează toate funcționalitățile legate de stocarea datelor local, asigurând păstrarea informațiilor utilizatorului, cum ar fi detalii personale și setări.

- **Managers:**

- Reprezintă partea logică a aplicației, gestionând funcționalități cheie:
 - * **SpeechManager**: Gestionarea funcționalităților de recunoaștere vocală sau citire text.
 - * **LocationManager**: Determinarea locației utilizatorului prin GPS, utilizată pentru a transmite informații către backend.
 - * **MotionManager**: Detectarea mișcărilor și impacturilor utilizând senzori disponibilului.
 - * **NotificationManager**: Gestionarea notificărilor locale, utilizate pentru alertarea utilizatorului în caz de urgență.
 - * **NetworkManager**: Gestionarea cererilor HTTP și comunicarea cu backend-ul.

- **View:**

- Include toate componentele grafice ale aplicației. Ecranele principale sunt:
 - * **LoginView**: Permite autentificarea utilizatorului.
 - * **RegisterView**: Ecranul pentru înregistrarea unui cont nou.
 - * **MainView**: Interfața principală a aplicației, unde utilizatorul poate naviga între funcțiile esențiale.
 - * **PatientFormView**: Formularul pacientului, unde se pot introduce și actualiza informațiile personale.
 - * **WelcomeView**: Ecranul inițial care ghidează utilizatorul spre înregistrare sau autentificare.

- **Tests:**

- Aplicația include module dedicate testării funcționalităților:
 - * **AsistentMobilTests**: Teste unitare pentru verificarea logicii aplicației.
 - * **AsistentMobilUITests**: Teste automate pentru verificarea interfeței grafice.

- **Assets și Info:**

- Activele grafice (imagini, iconițe) sunt organizate în folderul **Assets**.
- Fișierul **Info** conține metadate și configurații importante pentru aplicație.

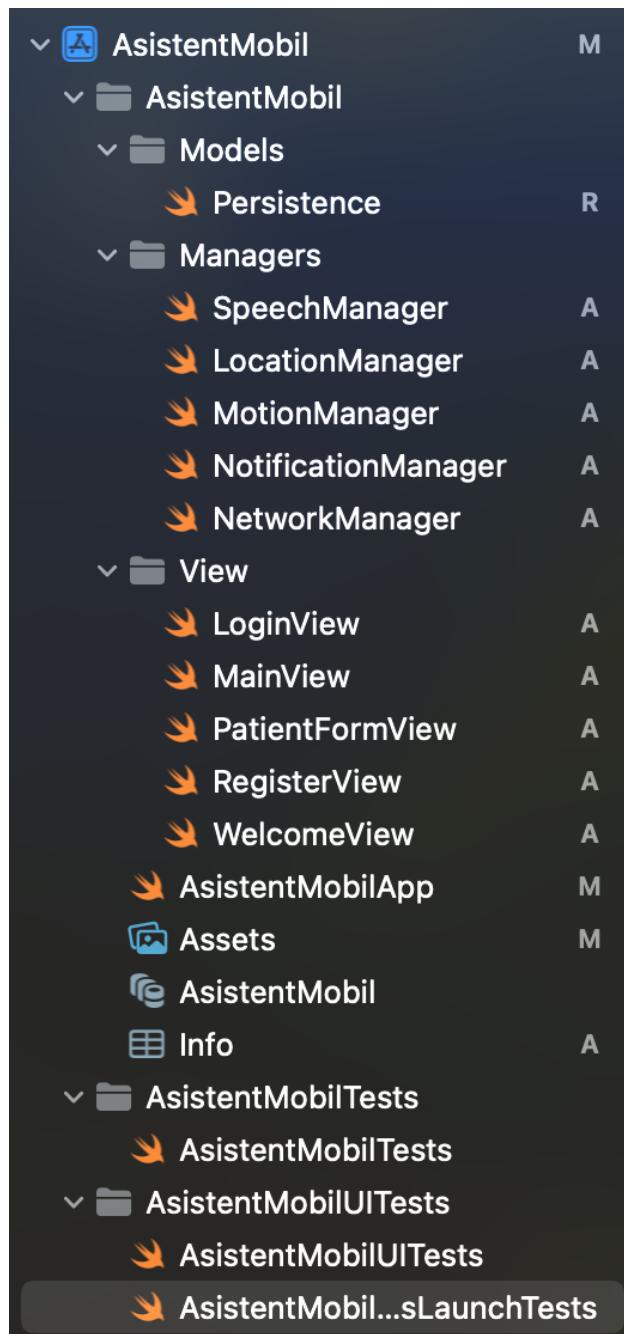


Figure 3.1: Structura proiectului aplicăției mobile Asistent Mobil.

Justificarea alegerii acestei arhitecturi Arhitectura modulară aduce multiple avantaje:

- **Separarea responsabilităților:** Fiecare modul gestionează o singură funcționalitate, ceea ce face codul mai ușor de înțeles și de întreținut.
- **Extensibilitate:** Aplicația poate fi extinsă cu noi funcționalități fără a afecta modulele existente.
- **Testabilitate:** Separarea logicii de interfață și organizarea codului facilitează implementarea și rularea testelor unitare și funcționale.
- **Scalabilitate:** Designul modular permite echipei de dezvoltare să colaboreze eficient, fiecare membru putând lucra pe module independente.

3.1.1.2 2. Configurarea proiectului în Xcode

Proiectul a fost creat în Xcode utilizând template-ul **App** pentru Swift. Configurațiile inițiale includ:

- **Limbaj:** Swift.
- **Framework-uri utilizate:**
 - **CoreMotion:** pentru accesarea senzorilor (accelerometru).
 - **UIKit:** pentru crearea interfeței grafice a aplicatiei.
 - **UserNotifications:** pentru gestionarea notificărilor locale.
- **Versiune minimă iOS:** iOS 13+, pentru a asigura compatibilitatea cu majoritatea dispozitivelor active.

3.1.1.3 Biblioteci și framework-uri utilizate

1. CoreMotion:

- Utilizată pentru a accesa datele senzorilor dispozitivului (accelerometru, giroscop).
- A permis detectarea mișcărilor bruște și analiza datelor brute pentru identificarea impacturilor.

2. UserNotifications:

- Gestionarea notificărilor locale pentru alertarea utilizatorilor.
- Configurarea notificărilor personalizate pentru evenimente critice.

3.1.1.4 Configurarea permisiunilor

Pentru a accesa funcționalitățile necesare, aplicația a solicitat permisiuni de la utilizator. Acestea au fost configurate în fișierul Info.plist, adăugând următoarele chei și descrieri:

- **NSMicrophoneUsageDescription:** Aplicația utilizează microfonul pentru a detecta răspunsurile utilizatorului în caz de impact.
- **NSSpeechRecognitionUsageDescription:** Aplicația utilizează recunoașterea vocală pentru a detecta răspunsurile utilizatorului în caz de impact.
- **NSLocationWhenInUseUsageDescription:** Avem nevoie de locația dumneavoastră pentru a detecta accidente.
- **NSLocationAlwaysUsageDescription:** Avem nevoie de locația dumneavoastră pentru a detecta accidente, chiar și când aplicația rulează în fundal.

Figura de mai jos ilustrează modul în care aceste permisiuni sunt configurate în fișierul Info.plist.

Key	Type	Value
Information Property List	Dictionary	(10 items)
> App Transport Security Settings	Dictionary	(1 item)
Launch screen interface file base name	String	LaunchScreen
Bundle name	String	AsistentMobil
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	cim.lacinschiAnda.AsistenteMobil
Bundle display name	String	AsistentMobil
Privacy - Location Always Usage Description	String	Utilizăm locația pentru a detecta accidente chiar și când aplicația rulează în fundal.
Privacy - Location When In Use Usage Description	String	Utilizăm locația pentru a detecta accidente.
Privacy - Microphone Usage Description	String	Aplicația utilizează microfonul pentru a detecta răspunsurile utilizatorului în caz de impact.
Privacy - Speech Recognition Usage Description	String	Aplicația utilizează recunoașterea vocală pentru a detecta răspunsurile utilizatorului în caz de impact.

Figure 3.2: Configurarea permisiunilor în fisierul Info.plist.

Aceste permisiuni sunt esențiale pentru asigurarea funcționalității de bază a aplicației și respectarea standardelor de confidențialitate și securitate a datelor utilizatorilor.

3.1.1.5 Configurarea inițială a aplicației

Această configurație inițială din aplicația **AsistentMobilApp** pregătește mediul pentru interacțiunile principale cu utilizatorul. Este utilizat un manager vocal (**SpeechManager**) pentru detectarea comenziilor vocale și un manager de mișcare (**MotionManager**) pentru monitorizarea mișcărilor utilizatorului, ceea ce sugerează un sistem interactiv bazat pe senzori. De asemenea, include un mecanism pentru autentificarea utilizatorilor printr-un token salvat în aplicație, gestionând starea de conectare.

```

AsistentMobil > AsistentMobil > AsistentMobilApp > No Selection
1 import SwiftUI
2 import UserNotifications
3
4 @main
5 struct AsistentMobilApp: App {
6     // Manager pentru recunoașterea vocală
7     @StateObject private var speechManager = SpeechManager()
8     // Manager pentru detectarea mișcărilor
9     @StateObject private var motionManager = MotionManager(speechManager: speechManager())
10    // Stare care arată dacă utilizatorul este autenticat
11    @State private var isAuthenticated: Bool = false
12    // Tokenul de autentificare salvat în aplicație
13    @AppStorage("authToken") var authToken: String = ""
14
15    // Resetez token-ul de autentificare când aplicația pornește
16    init() {
17        UserDefaults.standard.removeObject(forKey: "authToken")
18    }
19
20    var body: some Scene {
21        WindowGroup {
22            // Dacă utilizatorul este autenticat, se afișează ecranul principal
23            if isAuthenticated || !authToken.isEmpty {
24                MainView(isAuthenticated: $isAuthenticated, motionManager: motionManager, speechManager: speechManager)
25            } else {
26                // Dacă nu, se afișează ecranul de bun venit
27                WelcomeView(isAuthenticated: $isAuthenticated)
28            }
29        }
30    }
31 }
32
33 // Delegatul aplicației pentru evenimentele principale
34 class AppDelegate: NSObject, UIApplicationDelegate {
35     func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
36         // Cere permisiuni pentru notificări
37         UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) { granted, error in
38             if let error = error {
39                 print("Eroare la cererea permisiunilor pentru notificări: \(error.localizedDescription)")
40             }
41         }
42         return true
43     }
}

```

Figure 3.3: Configurarea inițială a aplicației în fisierul

Mai mult, aplicația resetează token-ul la fiecare pornire pentru securitate sporită. În funcție de starea de autentificare, utilizatorului îi este afișat fie ecranul principal, fie cel de bun venit. Prin integrarea notificărilor și cererea permisiunilor necesare de la utilizator, se asigură o comunicare optimă între aplicație și utilizator. Aceste setări creează baza funcționalităților aplicației, pregătind-o pentru utilizarea ulterioară.

3.1.1.6 Implementarea funcționalităților

În acest subcapitol, voi descrie pas cu pas implementarea funcționalităților critice ale aplicației mobile. Scopul este să prezint modul în care fiecare componentă contribuie la funcționalitatea globală, să explic fluxurile de date și să ofer detalii tehnice.

1. Detectarea căderilor

Detectarea căderilor este un punct central al aplicației, necesitând o implementare tehnică detaliată și bine calibrată pentru a asigura fiabilitate și acuratețe. Această funcționalitate implică utilizarea accelerometrului pentru a analiza mișările dispozitivului și pentru a detecta impacturi bruse asociațe cu căderi. Vom explora în detaliu fiecare pas, de la teoria din spatele calculului magnitudinii accelerării până la implementarea practică și optimizarea algoritmului.

Accelerometru și măsurarea accelerării

Un accelerometru măsoară accelerarea liniară pe cele trei axe ortogonale ale dispozitivului: **X**, **Y** și **Z**. Datele brute furnizate de accelerometru reprezintă accelerarea totală, care include atât accelerarea gravitațională de **1.0 g** (corespunzătoare **9.8 m/s²**), cât și accelerările datorate mișărilor utilizatorului. Într-un dispozitiv staționar, valoarea **Z** va reflecta aproximativ **1.0 g**, în timp ce valorile pentru **X** și **Y** vor fi apropiate de zero.

Formula pentru calculul magnitudinii accelerării

Magnitudinea accelerării este utilizată pentru a evalua intensitatea mișării. Aceasta este determinată folosind următoarea formulă:

$$\text{Magnitudine} = \sqrt{X^2 + Y^2 + Z^2}$$

Explicație:

- Valorile **X**, **Y**, și **Z** reprezintă accelerarea pe fiecare axă.
- Formula calculează rezultanta vectorială a accelerării pe cele trei axe, oferind o măsură totală a intensității mișării.

Exemplu practic:

Dacă $X = 0.5\text{ g}$, $Y = 0.6\text{ g}$, și $Z = 1.0\text{ g}$:

$$\text{Magnitudine} = \sqrt{(0.5)^2 + (0.6)^2 + (1.0)^2} = \sqrt{0.25 + 0.36 + 1.0} = \sqrt{1.61} \approx 1.27\text{ g}$$

Această formulă este baza algoritmului utilizat pentru detectarea căderilor, oferind o măsură simplă și eficientă pentru analiza mișărilor dispozitivului.

Interpretarea magnitudinii accelerării

- **Staționar (mișcare minimă):** Dacă dispozitivul este nemișcat, magnitudinea accelerării va fi aproape de 1.0 g , datorită gravitației.
- **Mișări normale:** Activitățile zilnice, precum mersul sau urcatul scărilor, produc valori între 0.5 g și 2.0 g .
- **Căderi bruse:** În timpul unei căderi, magnitudinea accelerării crește rapid, adesea depășind 2.5 g , în funcție de severitatea impactului.

Determinarea pragului optim

Pragul optim pentru detectarea căderilor este determinat prin testare practică. Pe baza studiilor și scenariilor testate:

- **Magnitudini tipice:**

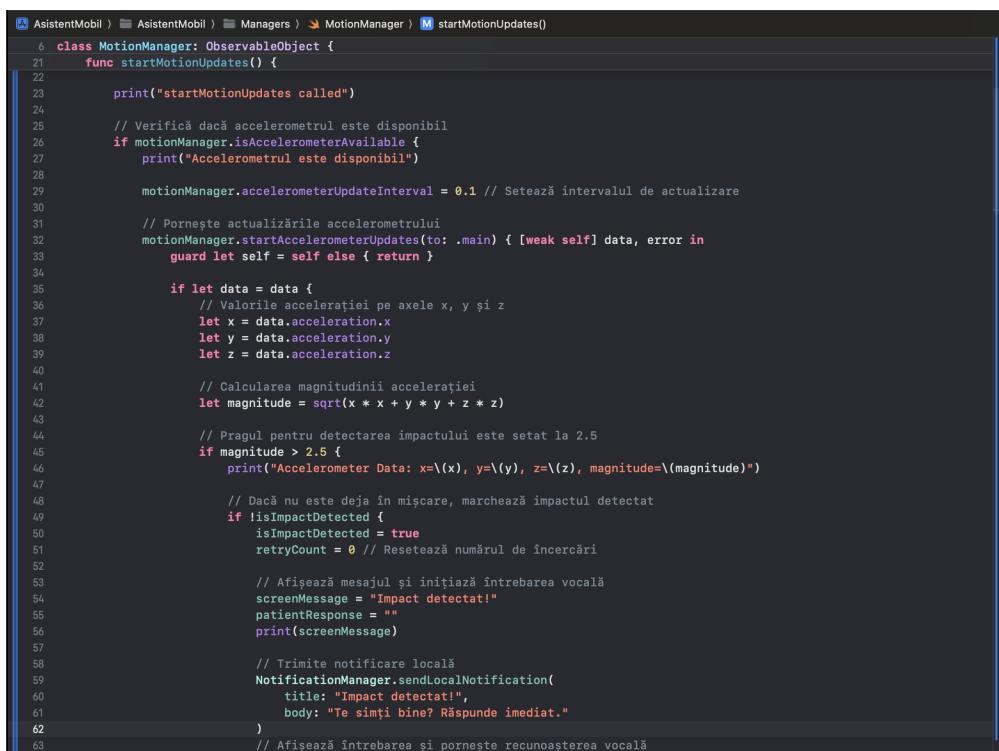
- Mișcări normale: $0.5 g - 1.5 g$
- Căderi: $2.0 g - 4.0 g$

- **Prag recomandat:** $2.5 g$

- Este un punct de referință care echilibrează sensibilitatea și specificitatea.
- Reduce alertele false fără a omite detectarea căderilor reale.

Implementarea în aplicație

În aplicație, detectarea căderilor este realizată prin utilizarea accelerometrului dispozitivului, care măsoară accelerația pe cele trei axe. Magnitudinea accelerării este calculată pentru a evalua intensitatea mișcării și a determina dacă aceasta depășește pragul stabilit pentru detectarea unei căderi.



```

AsistentMobil ) AsistentMobil ) Managers ) MotionManager ) startMotionUpdates()
6 class MotionManager: ObservableObject {
7     func startMotionUpdates() {
8
9         print("startMotionUpdates called")
10
11         // Verifică dacă accelerometrul este disponibil
12         if motionManager.isAccelerometerAvailable {
13             print("Accelerometrul este disponibil!")
14
15             motionManager.accelerometerUpdateInterval = 0.1 // Setează intervalul de actualizare
16
17             // Pornește actualizările accelerometrului
18             motionManager.startAccelerometerUpdates(to: .main) { [weak self] data, error in
19                 guard let self = self else { return }
20
21                 if let data = data {
22                     // Valorile accelerării pe axele x, y și z
23                     let x = data.acceleration.x
24                     let y = data.acceleration.y
25                     let z = data.acceleration.z
26
27                     // Calcularea magnitudinii accelerării
28                     let magnitude = sqrt(x * x + y * y + z * z)
29
30                     // Pragul pentru detectarea impactului este setat la 2.5
31                     if magnitude > 2.5 {
32                         print("Accelerometer Data: x=\(x), y=\(y), z=\(z), magnitude=\(magnitude)")
33
34                         // Dacă nu este deja în mișcare, marchează impactul detectat
35                         if !isImpactDetected {
36                             isImpactDetected = true
37                             retryCount = 0 // Resetează numărul de încercări
38
39                             // Afisează mesajul și inițiază întrebarea vocală
40                             screenMessage = "Impact detectat!"
41                             patientResponse = ""
42                             print(screenMessage)
43
44                             // Trimite notificare locală
45                             NotificationManager.sendLocalNotification(
46                                 title: "Impact detectat!",
47                                 body: "Te simți bine? Răspunde imediat."
48                             )
49                         // Afisează întrebarea și pornește recunoașterea vocală
50                     }
51                 }
52             }
53         }
54     }
55 }

```

Figure 3.4: Codul implementat pentru detectarea impactului

Codul descris mai sus preia datele accelerometrului, calculează magnitudinea și o compară cu pragul stabilit pentru a detecta dacă a avut loc un impact. Dacă magnitudinea depășește pragul de **2.5 g**, se consideră că a avut loc o cădere și se declanșează notificarea vocală pentru utilizator.

Prin utilizarea framework-ului CoreMotion din iOS, accelerometrul colectează valorile de accelerare pe cele trei axe **X**, **Y** și **Z** ale dispozitivului. Aceste date sunt folosite pentru a calcula magnitudinea mișcării și a decide dacă aceasta reprezintă o cădere.

Gestionarea erorilor: Codul include și o gestionare a erorilor care poate apărea în cazul în care accelerometrul nu este disponibil sau dacă există alte probleme la preluarea datelor de la senzor. Astfel, erorile sunt gestionate pentru a preveni blocarea aplicației.

În final, sistemul permite oprirea actualizărilor accelerometrului atunci când nu mai sunt necesare, economisind astfel resursele dispozitivului. Acest lucru este realizat cu ajutorul metodei ‘*stopAccelerometerUpdates*‘, care asigură că procesul de monitorizare este oprit corespunzător atunci când aplicația nu mai este activă.

2. Notificările locale și gestionarea alertelor

În cadrul aplicației, notificările locale joacă un rol esențial în alertarea utilizatorului imediat ce un impact semnificativ este detectat. Notificările sunt trimise utilizatorului pentru a-l informa rapid și eficient despre un posibil accident.

1. Solicitarea permisiunii pentru notificări

Pentru ca aplicația să poată trimite notificări, trebuie să obțină permisiunea utilizatorului. Acest lucru se face utilizând framework-ul **UserNotifications** din iOS. În codul tău, permisiunea pentru notificări este solicitată atunci când aplicația este lansată.

Codul pentru solicitarea permisiunii de a trimite notificări este:

```
func requestNotificationPermission() {
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) { granted, error in
        if let error = error {
            print("Notification permission error: \(error.localizedDescription)")
        } else if granted {
            print("Notification permission granted")
        } else {
            print("Notification permission denied")
        }
    }
}
```

Figure 3.5: Codul implementat pentru detectarea impactului

Explicație:

- **requestAuthorization(options:)**: Cere permisiuni pentru a trimite notificări, inclusiv alerte vizuale, sunete și badge-uri.
- **Gestionarea permisiunilor**: Dacă permisiunile sunt acordate, aplicația va putea trimite notificări; dacă sunt refuzate, aplicația nu va putea să le utilizeze.

2. Crearea notificărilor locale

După ce permisiunea este acordată, notificările pot fi trimise utilizatorului pentru a-l alerta în cazul unui impact. Aceste notificări pot conține un **titlu**, **mesajul** impactului, și **sunet**. De asemenea, notificările pot fi programate să se declanșeze la un interval de timp specificat, folosind un **trigger**.

Codul pentru trimiterea notificării este:

```
// Clasa pentru gestionarea notificărilor locale
class NotificationManager {
    static func sendLocalNotification(title: String, body: String) {
        // Creează conținutul notificării
        let content = UNMutableNotificationContent()
        content.title = title
        content.body = body
        content.sound = .default // Sunet implicit pentru notificare

        // Setează triggerul notificării (după 1 secundă)
        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)

        // Creează cererea de notificare
        let request = UNNotificationRequest(
            identifier: UUID().uuidString, // Identificator unic
            content: content,
            trigger: trigger
        )

        // Adaugă notificarea la centrul de notificări
        UNUserNotificationCenter.current().add(request) { error in
            if let error = error {
                print("Eroare la trimiterea notificării: \(error.localizedDescription)")
            }
        }
    }
}
```

Figure 3.6: Codul implementat pentru detectarea impactului

Explicație:

- **UNMutableNotificationContent**: Creează o notificare cu un titlu, mesaj și sunet.
- **UNTimeIntervalNotificationTrigger**: Declanșează notificarea după un interval de timp specificat (în acest caz, 1 secundă după detectarea impactului).
- **UNNotificationRequest**: Este folosit pentru a crea cererea de notificare care va fi trimisă către **UserNotificationCenter**.
- **add(request)**: Trimită notificarea la coada de notificări a aplicației.

3. Trimiterea datelor către server

În cadrul aplicației, trimiterea datelor relevante despre accident către un server este un pas esențial pentru procesarea și stocarea corectă a informațiilor, precum locația utilizatorului și detalii despre impact. Aceste date sunt transmise către server pentru a fi prelucrate și gestionate corespunzător, permitând intervenții rapide și eficiente.

Crearea și configurarea cererii HTTP

Primul pas în trimiterea datelor către server este crearea unui dicționar care conține informațiile relevante despre accident. Acesta include locația utilizatorului, detalii despre impact și starea pacientului. După ce datele sunt colectate, ele sunt serializate în format JSON, iar cererea HTTP este configurată pentru a trimită aceste informații către server.

Codul pentru configurarea cererii și trimiterea datelor către server este următorul:

```

func sendAccidentDataToServer() {
    guard let url = URL(string: "http://192.168.100.11:5000/api/report-accident") else {
        print("URL invalid")
        return
    }
    guard let token = UserDefaults.standard.string(forKey: "authToken"), !token.isEmpty else {
        print("User not authenticated")
        return
    }
    guard let location = LocationManager.shared.location else {
        print("Locația nu este disponibilă")
        return
    }
    let accidentData: [String: Any] = [
        "location": [
            "latitude": location.coordinate.latitude,
            "longitude": location.coordinate.longitude
        ],
        "details": "Impact detected",
        "time": ISO8601DateFormatter().string(from: Date()),
        "stare": patientResponse
    ]
    guard let httpBody = try? JSONSerialization.data(withJSONObject: accidentData, options: []) else {
        print("Eroare la serializarea JSON")
        return
    }
    var request = URLRequest(url: url)
    request.httpMethod = "POST"
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    request.httpBody = httpBody

    URLSession.shared.dataTask(with: request) { data, response, error in
        if let error = error {
            print("Eroare la trimiterea datelor: \(error.localizedDescription)")
            return
        }
        guard let response = response as? HTTPURLResponse, response.statusCode == 200 else {
            print("Serverul a returnat o eroare")
            return
        }
        print("Datele au fost trimise cu succes!")
    }
}

```

Figure 3.7: Codul pentru crearea cererii de trimitere a datelor către server

Explicație detaliată a codului:

- **Verificarea URL-ului și a autentificării:** Codul începe prin verificarea validității URL-ului (`http://192.168.100.11:5000/api/report-accident`). Dacă URL-ul nu este valid, funcția se oprește. Apoi, se verifică dacă token-ul de autentificare (`authToken`) este disponibil. Dacă lipsește, utilizatorul nu este autentificat, iar execuția este oprită.
- **Obținerea locației utilizatorului:** Se utilizează `LocationManager.shared.location` pentru a accesa coordonatele GPS. Dacă locația nu este disponibilă, se afișează un mesaj de eroare și execuția se oprește.
- **Crearea unui dicționar cu datele accidentului:** Un dicționar numit `accidentData` este creat și conține următoarele informații:
 - **Locația:** coordonatele GPS (latitudine și longitudine).
 - **Detalii:** un mesaj standard "Impact detected" care indică detectarea unui accident.
 - **Ora:** timpul curent în format ISO8601.
 - **Starea pacientului:** preluată din variabila `patientResponse`.
- **Serializarea datelor în format JSON:** Dicționarul `accidentData` este serializat folosind `JSONSerialization.data (withJSONObject:)` pentru a-l transforma într-un corp de cerere (`httpBody`) compatibil cu formatul JSON.
- **Crearea cererii HTTP:** Se inițiază un obiect `URLRequest` configurat pentru metoda POST. Sunt adăugate headere HTTP: `Content-Type` este setat la `application/json` pentru a specifica formatul datelor, iar `Authorization` include token-ul de autentificare.

- **Trimiterea cererii:** Cererea este trimisă utilizând URLSession.shared.dataTask. Dacă există erori în procesul de trimitere sau dacă răspunsul serverului nu indică succes (statusCode == 200), se afișează mesaje de eroare corespunzătoare.
- **Mesaj de succes:** Dacă cererea este procesată cu succes, se afișează mesajul "Datele au fost trimise cu succes!".

Gestionarea erorilor

Într-un sistem în care datele sunt trimise către un server, gestionarea erorilor este crucială. Dacă există o eroare de rețea sau serverul returnează un cod de eroare, aplicația va captura și loga detaliile pentru a fi rezolvate rapid.

- Dacă apare o eroare la trimitera cererii, aceasta este capturată și se afișează un mesaj detaliat.
- Dacă serverul returnează un cod de eroare (altul decât 200 OK), aplicația va loga eroarea respectivă, iar procesul se va opri.

Acest mecanism asigură că aplicația funcționează corect chiar și în fața unor eventuale probleme de rețea sau server.

Autentificarea utilizatorului

Pentru a asigura securitatea aplicației, trimitera datelor către server se face doar dacă utilizatorul este autentificat. Acest lucru se realizează prin utilizarea unui **token de autentificare**, care este stocat local în aplicație. Token-ul este obținut atunci când utilizatorul se autentifică și este folosit pentru a valida cererile trimise către server. Astfel, serverul poate verifica că cererea provine de la un utilizator autenticat.

Prin implementarea acestei funcționalități, aplicația permite trimitera de date importante despre accident către server, asigurând o procesare rapidă și corectă. Verificările de autentificare, locație și validare a datelor sunt realizate înainte de a trimite informațiile. Serverul primește datele într-un format JSON, iar aplicația poate detecta și gestiona erorile eficient, oferind feedback utilizatorului în caz de succes sau eșec. Toate aceste procese sunt monitorizate prin loguri de succes și eroare, ceea ce permite o depanare ușoară și o experiență de utilizare fiabilă.

3. Interfața utilizatorului (UI)

În cadrul aplicației, interfața utilizatorului joacă un rol esențial în crearea unei experiențe de utilizare intuitive și eficiente. Aplicația este construită folosind framework-ul **UIKit**, care permite crearea unor interfețe grafice dinamice și atractive. Designul este centrat pe utilizator, având un flux simplu și ușor de înțeles pentru utilizatorii care trebuie să interacționeze cu aplicația în situații de urgență.

Structura aplicației și navigarea între ecrane

Structura aplicației este construită pentru a oferi o navigare fluidă între ecranele principale ale aplicației, cum ar fi „Delogare” și „Fișa Pacientului”. Aceste ecrane sunt integrate într-un **NavigationView**, care facilitează schimbul de informații între diferențele secțiuni ale aplicației.

Navigarea între ecrane

Navigarea între ecrane este gestionată folosind un **NavigationView**, care include funcționalitatea de a adăuga butoane pentru fiecare ecran important. De exemplu, ecranul principal oferă utilizatorului două opțiuni principale: **Delegare** și **Fișa Pacientului**.

Butonul „Delegare”

Butonul **Delegare** permite utilizatorului să iasă din contul activ și să revină la ecranul principal. Aceasta este esențial pentru gestionarea sesiunilor utilizatorilor.

Butonul „Fișa Pacientului”

Butonul **Fișa Pacientului** permite utilizatorului să acceseze informațiile pacientului, oferind detalii esențiale precum numele, datele de contact și istoricul medical. Acesta este folosit pentru a vizualiza și actualiza informațiile pacientului, fiind o componentă centrală a aplicației.



Figure 3.8: Ecranul principal al aplicației, cu opțiunile de „Delegare” și „Fișa Pacientului”

2. Crearea și completarea fișei pacientului

Fisa pacientului este o secțiune crucială a aplicației, oferind informații detaliate despre starea pacientului și istoricul său medical. Utilizatorul poate vizualiza, edita și actualiza aceste informații în orice moment.



Figure 3.9: Ecranul „Fișa Pacientului” în care sunt prezentate detaliile pacientului

Fișa pacientului include câmpuri pentru:

- Numele pacientului:** Câmpuri pentru numele complet.
- Datele de contact:** Adresa, telefonul și alte informații utile.
- Istoricul medical:** Detalii despre afecțiuni anterioare sau tratamente.
- Medicamentele:** Lista de medicamente și tratamente actuale.

După ce informațiile sunt complete sau actualizate, utilizatorul poate salva modificările apasând pe butonul **Actualizează Informațiile**.



Figure 3.10: Modificarea și actualizarea informațiilor pacientului în „Fișa Pacientului”

3. Ecranul de înregistrare și autentificare

În cadrul aplicației, utilizatorii pot crea un cont nou sau se pot autentifica pentru a accesa funcționalitățile aplicației. Ecranul de înregistrare permite utilizatorilor să se înscrie pentru un cont nou, în timp ce ecranul de autentificare este utilizat pentru a se conecta la un cont existent.

Ecranul de înregistrare

Pe ecranul de înregistrare, utilizatorii trebuie să completeze câmpuri pentru a-și introduce numele, emailul și parola. După completarea acestor câmpuri, utilizatorul poate apăsa butonul **Înregistrare** pentru a crea un cont nou.



Figure 3.11: Ecranul „Înregistrare” al aplicației pentru crearea unui cont

Ecranul de autentificare

După înregistrare, utilizatorii pot să se conecteze la aplicație prin furnizarea unui email și unei parole deja create. Apăsând butonul **Logare**, utilizatorul va fi autentificat și va avea acces la funcționalitățile aplicatiei.



Figure 3.12: Ecranul de alegere între „Logare” și „Înregistrare”

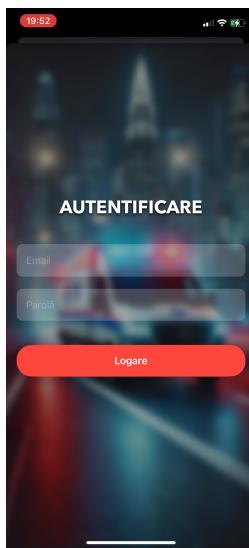


Figure 3.13: Ecranul „Autentificare” pentru utilizatorii existenți

4. Designul general al aplicației

Designul aplicației este realizat astfel încât să ofere o experiență de utilizare optimizată. Ecranele sunt clare și ușor de navigat, iar butoanele sunt plasate în locații accesibile pentru utilizatorii cu mobilitate limitată.

Stilul vizual

Stilul aplicației include un fundal simplu și un contrast puternic între text și elementele interactive pentru a asigura lizibilitatea. De asemenea, culorile sunt alese pentru a transmite calm și siguranță, un aspect esențial pentru o aplicație de urgență.

Fluiditatea navigării

Navigarea între ecrane este rapidă și intuitivă. Folosind **NavigationView**, aplicația permite utilizatorilor să se deplaseze ușor între diferitele secțiuni ale aplicației, iar fiecare secțiune are un scop clar și bine definit, ceea ce reduce timpul de învățare al utilizatorilor.

Interfața utilizatorului din această aplicație este gândită astfel încât să fie cât mai accesibilă și ușor de utilizat, chiar și în condiții de stres. Navigarea între secțiuni este simplă, iar designul facilitează completarea rapidă a acțiunilor necesare. Prin utilizarea unor componente standardizate și prin adăugarea unui flux logic, aplicația reușește să ofere o experiență pozitivă utilizatorilor care au nevoie de asistență rapidă în caz de urgență.

5. Gestionaarea notificărilor și a interacțiunii cu utilizatorul

Această secțiune descrie detaliat logica implementată în cadrul aplicației pentru gestionarea notificărilor și interacțiunii cu utilizatorul în situații de urgență, precum detectarea unui impact. Este abordată afișarea notificărilor, interacțiunea vocală și notificarea dispeceratului în funcție de răspunsurile primite.

5.1. Detectarea impactului

Atunci când aplicația detectează un impact, acesta este semnalizat imediat printr-o notificare vizuală afișată pe ecranul dispozitivului utilizatorului. Această notificare reprezintă prima etapă a procesului de alertare.

Figura de mai jos prezintă mesajul afișat utilizatorului după detectarea impactului:



Figure 3.14: *Mesaj initial de notificare a impactului*

În plus, aplicația trimită o notificare push către utilizator pentru a-i aduce la cunoștință evenimentul detectat.



Figure 3.15: *Notificarea trimisă utilizatorului la detectarea impactului*

5.2. Interacțiunea vocală cu utilizatorul

După notificarea impactului, aplicația initiază o interacțiune vocală prin adresarea întrebării: „**Te simți bine?**”. Acest mecanism asigură colectarea unui răspuns din partea utilizatorului și identifi-

carea stării sale fizice.



Figure 3.16: Întrebarea vocală adresată utilizatorului

Fluxul interacțiunii este structurat astfel:

1. Dacă utilizatorul răspunde „Da”, aplicația afișează mesajul: „Pacientul se simte bine” și continuă monitorizarea în fundal.



Figure 3.17: Mesajul afișat în cazul unui răspuns pozitiv

2. Dacă utilizatorul răspunde „Nu”, aplicația afișează mesajul: „Pacientul nu se simte bine, are nevoie de o ambulanță” și notifică automat dispeceratul despre urgență.



Figure 3.18: Mesajul afișat în cazul unui răspuns negativ

3. Dacă utilizatorul nu răspunde în interval de 10 secunde, aplicația reiterează întrebarea pentru a-i oferi o a doua șansă de răspuns. Dacă nici de această dată nu răspunde, aplicația afișează mesajul: „*Pacientul nu răspunde*”.



Figure 3.19: Mesajul în care pacientul este întrebat a doua oară.

5.3. Gestionarea lipsei de răspuns

Dacă utilizatorul nu răspunde nici după a doua încercare de interacțiune vocală, aplicația gestionează situația astfel:

1. Dacă utilizatorul răspunde „Da” în urma repetării întrebării, aplicația afișează mesajul „*Pacientul se simte bine*” și notifică dispeceratul că utilizatorul nu se află în pericol.
2. Dacă utilizatorul răspunde „Nu” în urma repetării, aplicația afișează mesajul „*Pacientul nu se simte bine, are nevoie de o ambulanță*” și notifică dispeceratul despre situația de urgență.
3. Dacă utilizatorul nu răspunde nici la a doua întrebare, aplicația afișează mesajul „*Pacientul nu răspunde*” și informează dispeceratul despre lipsa de răspuns din partea utilizatorului.



Figure 3.20: Mesajul afișat în cazul lipsei de răspuns din partea utilizatorului

Această secvență logică este proiectată pentru a acoperi toate cazurile posibile, asigurând o notificare rapidă și precisă a dispeceratului în funcție de răspunsurile utilizatorului sau de lipsa acestora.

3.2 Backend

3.2.1 Tehnologii utilizate

Backend-ul aplicației a fost construit folosind **Flask**, un framework ușor și flexibil pentru Python, care permite dezvoltarea rapidă a aplicațiilor web. Am utilizat mai multe pachete și module pentru a adresa cerințele aplicației:

- **Flask**: Framework-ul principal pentru crearea aplicației web și gestionarea cererilor HTTP.
- **Flask-SQLAlchemy**: Oferă integrarea facilă a bazei de date SQL cu Flask, gestionând modelele și operațiile de bază de date.
- **Flask-SocketIO**: Folosit pentru a adăuga funcționalități WebSocket, permitând aplicației să trimită notificări în timp real către utilizatori.
- **JWT (JSON Web Tokens)**: Utilizat pentru autentificarea utilizatorilor și protecția cererilor API.

3.2.2 Structura backend-ului

Backend-ul este responsabil pentru procesarea cererilor HTTP, gestionarea sesiunilor utilizatorilor și a datelor relevante (cum ar fi accidentul și locația). Arhitectura este simplă și modulară, folosind un framework robust care facilitează dezvoltarea rapidă a API-urilor.

1. Rute și Endpoints

- **POST /api/report-accident:** Această rută primește și salvează informațiile despre accidentele raportate.
- **GET /api/status:** O rută de verificare a stării aplicației.
- **POST /api/login:** Răspunde pentru autentificarea utilizatorilor și generarea unui token JWT.
- **POST /api/register:** Permite înregistrarea unui utilizator nou.

2. Autentificarea utilizatorilor

- Folosind **JWT**, utilizatorii se pot autentifica prin furnizarea unui nume de utilizator și a unei parole. După autentificare, serverul generează un token care trebuie trimis în fiecare cerere ulterioară.

3. Gestionarea sesiunilor utilizatorilor

- **Session management:** Folosim sesiuni pentru a stoca informațiile despre utilizatorul conectat.

3.2.3 Răspunsuri și validarea datelor

Serverul validează datele primite pentru a asigura integritatea acestora înainte de a le salva în baza de date. În acest mod, datele despre accident și informațiile pacientului sunt verificate pentru corectitudine.

- **Validarea locației:** Dacă locația este incompletă sau invalidă, serverul va returna o eroare.
- **Verificarea autentificării:** Cererile de la utilizatori neautentificați vor fi respinse.
- **Mesaje de succes și eroare:**
 - Succes (cod 200): Când cererea este procesată cu succes, serverul trimitе un mesaj de succes.
 - Eroare (cod 400): Dacă cererea nu este validă (de exemplu, lipsesc date importante), serverul returnează un mesaj de eroare.

3.2.4 Securitatea backend-ului

Securitatea aplicației este o prioritate importantă, iar backend-ul este protejat prin mai multe măsuri de securitate:

- **Autentificare și autorizare cu JWT:** JWT este utilizat pentru a proteja rutele sensibile ale aplicației.
- **Criptarea parolelor:** Parolele utilizatorilor sunt criptate folosind algoritmi siguri, precum bcrypt.

3.2.5 Interacțiunea cu aplicația mobilă

Aplicația mobilă comunică cu backend-ul prin intermediul unui **REST API**. Aplicația trimite cereri HTTP pentru a obține date, a trimite informații despre accidente și pentru a obține actualizări în timp real.

- **Cererea de accident:** Aplicația trimite detalii despre accident, care sunt procesate de server și salvate în baza de date.
- **Autentificare și autorizare:** Aplicația trimite token-ul JWT în fiecare cerere pentru a verifica dacă utilizatorul este autentificat și poate accesa datele protejate.
- **Actualizări în timp real:** Aplicația folosește **WebSockets** pentru a primi notificări în timp real despre accidentele raportate sau alte actualizări relevante.

Backend-ul aplicației asigură gestionarea corectă și securizată a cererilor trimise de aplicația mobilă, validarea datelor și trimitera răspunsurilor către client. Cu ajutorul tehnologiilor moderne și al măsurilor de securitate, serverul procesează datele într-un mod eficient, asigurându-se că utilizatorii pot accesa informațiile și funcționalitățile aplicației într-un mod rapid și sigur.

3.3 Platforma web – Etape ale implementării

Platforma web dezvoltată în cadrul acestui proiect reprezintă un element central al sistemului, conectând aplicația mobilă și backend-ul pentru a oferi funcționalități esențiale atât pacienților, cât și personalului din dispecerat. Rolul său principal este de a facilita gestionarea datelor utilizatorilor și monitorizarea incidentelor într-un mod eficient și accesibil, contribuind astfel la siguranța și bunăstarea persoanelor cu probleme de mobilitate.

Pacienții pot utiliza platforma pentru a crea conturi, a se autentifica și a-și actualiza datele personale. Prin intermediul acesteia, utilizatorii pot introduce detalii relevante, precum informații medicale, contactele de urgență și locația curentă. Aceste informații sunt cruciale pentru a asigura un timp de răspuns minim în cazul unui incident. Simultan, personalul din dispecerat beneficiază de o interfață dedicată, care permite vizualizarea în timp real a alertelor și gestionarea eficientă a resurselor. Această abordare duală sprijină colaborarea dintre utilizatori și personalul specializat, contribuind la reducerea timpilor de reacție și la coordonarea eficientă a intervențiilor.

Integrarea notificărilor în timp real este una dintre cele mai importante funcționalități ale platformei. Atunci când aplicația mobilă detectează un incident sau o schimbare semnificativă, aceasta transmite o alertă către backend, care, la rândul său, o direcționează spre platforma web folosind tehnologia WebSockets. Acest mecanism asigură afișarea instantanee a alertelor pe tabloul de bord al dispeceratului, însăși de detalii precum locația utilizatorului și starea curentă. Prin urmare, platforma facilitează un timp de reacție redus și o intervenție bine informată.

Funcționalitățile principale ale platformei sunt structurate pentru a răspunde cerințelor ambelor categorii de utilizatori:

1. Pentru pacienți:

- **Înregistrare și autentificare:** Crearea unui cont nou care include introducerea de informații personale, precum numele, adresa și datele de contact. Sistemul de autentificare este securizat și utilizează token-uri pentru a proteja datele.
- **Actualizarea datelor:** Utilizatorii au posibilitatea de a-și modifica oricând informațiile personale, inclusiv datele medicale, contactele de urgență sau locația curentă.

2. Pentru personalul dispeceratului:

- **Monitorizarea alertelor:** Vizualizarea în timp real a incidentelor raportate, cu afişarea detaliilor critice, precum locaţia utilizatorului, timpul incidentului și starea acestuia.
- **Gestionarea utilizatorilor:** Accesul la informaţiile pacienţilor pentru a oferi asistenţă personalizată.
- **Tablou de bord centralizat:** Organizarea alertelor active și istoricul acestora într-un format clar și accesibil.

Din punct de vedere tehnic, platforma este construită pentru a integra tehnologii moderne care să asigure performanță și scalabilitate. Backend-ul dezvoltat cu Flask gestionează comunicarea dintre aplicația mobilă și platforma web, utilizând API-uri REST pentru preluarea și transmiterea datelor. Notificările în timp real sunt facilitate de WebSockets, ceea ce permite actualizarea imediată a datelor. Frontend-ul, dezvoltat cu React, oferă o interfață prietenoasă și responsivă, adaptată pentru diferite dimensiuni de ecran.

Platforma respectă standardele actuale de securitate, inclusiv criptarea datelor și autentificarea utilizatorilor. Sistemul de token-uri JWT protejează informațiile sensibile, iar toate comunicațiile sunt securizate prin protocolul HTTPS. Mai mult, fiecare acțiune este înregistrată, ceea ce oferă trasabilitate completă și conformitate cu normele GDPR.

Prin aceste funcționalități, platforma web joacă un rol important în coordonarea eficientă a intervențiilor și în crearea unui mediu sigur pentru utilizatori. Colaborarea fluidă dintre pacienți și personalul dispeceratului este realizată printr-o tehnologie robustă, concepută pentru a răspunde cerințelor complexe ale acestui domeniu.

3.3.1 Organizarea structurii proiectului

Platforma web a fost dezvoltată utilizând o arhitectură bine definită, care integrează tehnologii moderne pentru a asigura performanță, scalabilitate și ușurință în utilizare. Implementarea a fost împărțită între *frontend* (dezvoltat în React) și *backend* (dezvoltat în Flask), fiecare având un rol clar în sistem.

Structura proiectului:

Structura proiectului este organizată logic, separând componentele funcționale și logice pentru o dezvoltare și întreținere mai ușoară. Aceasta include directoare și fișiere care gestionează resursele statice, fișoarele HTML, scripturile backend și scripturile administrative. Structura generală este următoarea:

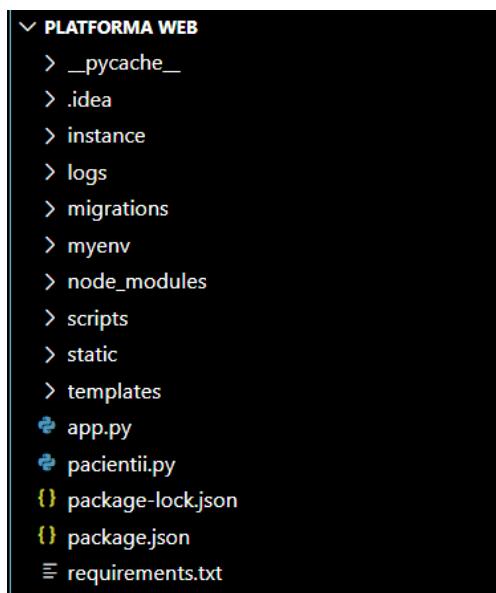


Figure 3.21: Structura proiectului web – organizarea directoarelor și fișierelor.

Descrierea componentelor principale:

- **instance:** Contine fișierul **users.db**, care stochează informațiile utilizatorilor și datele asociate lor. Acest director este esențial pentru gestionarea bazei de date.
- **migrations:** Include scripturi generate de Alembic pentru gestionarea modificărilor structurale ale bazei de date.
- **myenv:** Mediu virtual Python utilizat pentru a izola și organiza dependențele backend-ului.
- **logs:** Director utilizat pentru logurile de erori și alte evenimente importante. Exemple: fișierul **socket_errors.log**, care înregistrează problemele legate de notificările în timp real.
- **scripts:** Contine scripturi Python utilizate pentru administrarea proiectului:
 - **clear_alembic_version.py:** Curăță versiunile Alembic din baza de date.
 - **create_dispatcher_account.py:** Creează conturi pentru dispeceri.
 - **delete_temp_table.py:** Sterge tabelele temporare.
 - **reset_project.py:** Resetează proiectul la configurația inițială.
- **static:** Include resurse statice necesare pentru frontend:
 - **styles.css:** Stilizare.
 - **main.js:** Scripturi pentru funcționalități dinamice.
 - Imagini precum **ambulance_b.png**.
- **templates:** Include şablonane HTML folosite pentru generarea interfeței, cum ar fi:
 - **base.html:** Şablon principal utilizat pentru structura comună a paginilor.
 - **dashboard_dispatcher.html:** Dashboard-ul pentru dispeceri, care afișează alertele active și istoricul acestora.
 - **dashboard_patient.html:** Dashboard-ul pacienților, care permite vizualizarea și actualizarea datelor personale.

- *login_dispatcher.html*: Formular de autentificare pentru dispeceri.
 - *login_patient.html*: Formular de autentificare pentru pacienți.
 - *inregistrare_pacient.html*: Formular pentru crearea unui cont nou pentru pacienți.
 - *pacienti_logati.html*: Pagină care afișează lista pacienților autentificați.
 - *report_accident.html*: Formular pentru raportarea unui incident de către pacient.
- **app.py**: Fișierul principal care gestionează logica aplicației. Acesta include rutele backend, configurarea bazei de date și integrarea notificărilor în timp real.
 - **pacientii.py**: Script folosit pentru gestionarea datelor pacienților din baza de date.
 - **requirements.txt**: Conține lista dependențelor backend-ului utilizate în proiect, cum ar fi *Flask*, *SQLAlchemy* și *SocketIO*.

3.3.2 Configurarea mediului virtual

Platforma web a fost dezvoltată utilizând framework-ul Flask, datorită flexibilității sale, suportului extins pentru crearea API-urilor REST și a integrării simple cu alte module. Configurarea mediului și backend-ului a fost un proces etapizat, care a asigurat funcționalitatea corectă și extinderea ulterioară a aplicației. Fiecare pas este explicitat detaliat mai jos.

1. Crearea mediului virtual

Pentru a izola proiectul și a evita conflictele cu alte aplicații instalate pe sistem, s-a creat un mediu virtual Python. Acest pas este esențial în orice proiect profesional, deoarece permite utilizarea unor versiuni specifice de biblioteci fără a afecta alte proiecte.

Ce este un mediu virtual? Un mediu virtual este o copie izolată a instalării Python, care conține propriile fișiere executabile și librării. Aceasta permite instalarea pachetelor doar în cadrul proiectului curent, fără a afecta alte aplicații sau configurații globale.

2. Pașii pentru crearea mediului virtual

Crearea mediului virtual

Pentru a crea un mediu virtual, s-a utilizat modulul integrat *venv*. Comanda executată a fost următoarea:

```
python -m venv myenv
```

Aceasta a generat un director numit *myenv*, care conține toate fișierele necesare pentru rularea unui mediu virtual Python.

Activarea mediului virtual

După crearea mediului virtual, acesta a fost activat utilizând:

- **Windows:**

```
myenv\Scripts\activate
```

- **Linux/Mac:**

```
source myenv/bin/activate
```

După activare, prefixul (**myenv**) a apărut în terminal, indicând că toate comenziile ulterioare vor fi executate în acest mediu.

Dezactivarea mediului virtual

La finalul sesiunii de lucru, mediul virtual a fost dezactivat utilizând comanda: **deactivate**.

3. Instalarea pachetelor necesare

Pentru a implementa funcționalitățile backend-ului, au fost instalate următoarele pachete Python:

- **Flask:** Framework-ul principal utilizat pentru crearea aplicației backend.
- **Flask-SQLAlchemy:** Utilizat pentru gestionarea bazei de date și manipularea datelor.
- **Flask-Migrate:** Ajută la gestionarea migrărilor bazei de date (modificări ale tabelelor).
- **Flask-SocketIO:** Permite implementarea notificărilor în timp real folosind WebSockets.
- **Flask-JWT-Extended:** Oferă funcționalități avansate pentru autentificare și autorizare bazate pe token-uri JWT.

După activarea mediului virtual, toate dependențele proiectului au fost instalate folosind managerul de pachete **pip**. Pentru a menține consistența mediului de lucru, aceste dependențe au fost listate într-un fișier **requirements.txt**. Comanda utilizată a fost următoarea:

```
pip install -r requirements.txt
```

Figure 3.22: *Instalarea pachetelor necesare utilizând requirements.txt*.

Conținutul fișierului **requirements.txt**:

```
Flask==3.0.3
Flask-SQLAlchemy==3.1.1
Flask-Migrate==4.0.7
Flask-SocketIO==5.3.6
SQLAlchemy==2.0.32
python-socketio==5.11.3
python-engineio==4.9.1
requests==2.32.0
PyJWT==2.8.0
```

Figure 3.23: *Conținutul fișierului requirements.txt*.

Acest fișier asigură portabilitatea proiectului. Prin simpla rulare a comenzi **pip install -r requirements.txt**, mediul poate fi recreat pe orice alt sistem. Este important de menționat că pachetele prezentate mai sus reprezintă cele mai importante componente necesare funcționării backend-ului aplicației. Totuși, fișierul **requirements.txt** poate conține și alte pachete instalate

automat ca dependențe ale celor principale, cum ar fi **Werkzeug**, **itsdangerous**, **MarkupSafe**, sau **python-engineio**. Aceste pachete suportă funcționalitățile descrise și sunt gestionate implicit de managerul de pachete.

4. Avantajele utilizării unui mediu virtual

- **Izolare completă:** Toate bibliotecile sunt instalate local, în cadrul directorului ***myenv***, evitând interferențele cu alte proiecte.
- **Compatibilitate garantată:** Versiunile specifice ale bibliotecilor sunt controlate prin ***requirements.txt***, prevenind posibile erori cauzate de actualizări neprevăzute.
- **Portabilitate:** Configurația poate fi replicată cu ușurință pe alte sisteme folosind fișierul ***requirements.txt***.

3.3.3 Configurarea backend-ului

1. Configurarea fișierului principal (*app.py*)

Fișierul ***app.py*** reprezintă punctul central al aplicației backend, fiind responsabil pentru inițializarea aplicației, configurarea bazei de date, gestionarea notificărilor în timp real și definirea rutelor pentru interacțiunea utilizatorilor. Acest fișier adună toate componentele principale ale aplicației într-un singur loc, oferind un flux clar și eficient al datelor și proceselor.

Inițializarea aplicației Flask

Aplicația a fost construită utilizând framework-ul **Flask**, iar configurațiile principale includ setarea cheilor de securitate, conectarea la baza de date și integrarea notificărilor în timp real. Aceste configurații sunt esențiale pentru buna funcționare a aplicației și oferă suport pentru toate funcționalitățile backend.

```
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_socketio import SocketIO, disconnect
from flask import Flask

# Configurațiile aplicației
app.config['SECRET_KEY'] = 'your_secret_key' # Cheie pentru criptarea datelor
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db' # Conexiunea la baza de date
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False # Dezactivează notificările inutile

# Inițializarea bazei de date și a migrărilor

db = SQLAlchemy(app) # Gestionarea bazei de date
migrate = Migrate(app, db) # Gestionarea migrărilor bazei de date
socketio = SocketIO(app, cors_allowed_origins="*") # Notificări în timp real prin WebSockets
JWT_SECRET = 'your_jwt_secret_key'
```

Figure 3.24: Inițializarea aplicației Flask și configurarea componentelor principale.

Explicații detaliate despre configurări

- **SECRET_KEY:** Aceasta este o cheie criptografică utilizată pentru protecția datelor sensibile din aplicație, cum ar fi sesiunile, token-urile și alte informații critice. Este esențial ca această cheie să fie unică pentru fiecare aplicație și să fie păstrată secretă pentru a preveni accesul neautorizat. Aceasta joacă un rol crucial în securitatea generală a aplicației.
- **SQLALCHEMY_DATABASE_URI:** Această configurație definește locația bazei de date SQLite utilizate de aplicație. În cadrul proiectului, baza de date este salvată local, într-un fișier numit **users.db**. Alegerea SQLite oferă o soluție simplă și eficientă pentru aplicații de dimensiuni mici sau medii.
- **SQLALCHEMY_TRACK_MODIFICATIONS:** Configurată pe *False*, acest parametru previne consumul inutil de resurse asociat notificărilor despre modificările bazei de date. Optimizarea contribuie la îmbunătățirea performanței aplicației prin reducerea încărcării suplimentare asupra sistemului.
- **SQLAlchemy:** Este o extensie puternică folosită pentru gestionarea bazei de date. Aceasta permite definirea modelelor pentru tabele, manipularea datelor și realizarea interogărilor într-un mod eficient și intuitiv. SQLAlchemy este soluția standard în aplicațiile Flask pentru gestionarea bazelor de date relationale.
- **Migrate:** Extensie utilizată pentru gestionarea migrațiilor bazei de date. Aceasta permite crearea, modificarea și ștergerea tabelelor fără a afecta datele existente, asigurând consistența bazei de date pe parcursul evoluției aplicației.
- **SocketIO:** Flask-SocketIO facilitează notificările în timp real utilizând WebSockets. Aceasta permite transmiterea instantanea a informațiilor între server și client, îmbunătățind interacțiunea utilizatorilor. De asemenea, integrarea funcționalităților avansate pentru gestionarea alertelor și sincronizarea datelor asigură scalabilitatea și securitatea procesului de comunicare între componente.

Modelele bazei de date

Pentru gestionarea datelor aplicației, s-au definit modele folosind *SQLAlchemy*, un ORM (Object-Relational Mapper) puternic și flexibil. Fiecare model reprezintă o tabelă în baza de date și este utilizat pentru a interacționa cu datele într-un mod intuitiv și eficient.

- **Modelul Accident:** Acest model este utilizat pentru stocarea informațiilor despre incidente, cum ar fi locația (latitudine și longitudine), detaliile evenimentului și momentul în care a avut loc.

```
# Model pentru Accident
class Accident(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    latitude = db.Column(db.Float, nullable=False)
    longitude = db.Column(db.Float, nullable=False)
    details = db.Column(db.String(500), nullable=False)
    time = db.Column(db.DateTime, nullable=False)
```

Figure 3.25: Definirea modelului *Accident*.

- **Modelul Dispecerat:** Modelul stochează datele de autentificare ale dispecerilor. Fiecare dispecer are un nume de utilizator unic și o parolă securizată.

```
# Model pentru Dispecerat
class Dispatcher(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(150), nullable=False)
```

Figure 3.26: Definirea modelului Dispecerat.

- **Modelul Pacient:** Acest model este cel mai complex, stocând informații personale și medicale despre pacienți, precum și locația curentă. Această structură detaliată permite o gestionare eficientă a datelor utilizatorilor vulnerabili.

```
# Model pentru Pacient
class Patient(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(150), nullable=False)
    prenume = db.Column(db.String(150), nullable=True)
    nume = db.Column(db.String(150), nullable=True)
    data_nasterii = db.Column(db.Date, nullable=True)
    boala = db.Column(db.String(500), nullable=True)
    medicamente = db.Column(db.String(500), nullable=True)
    ultimul_incident = db.Column(db.String(500), nullable=True)
    tara = db.Column(db.String(100), nullable=True)
    judet = db.Column(db.String(100), nullable=True)
    oras = db.Column(db.String(100), nullable=True)
    strada = db.Column(db.String(200), nullable=True)
    numar = db.Column(db.String(50), nullable=True)
    cod_postal = db.Column(db.String(20), nullable=True)
    telefon = db.Column(db.String(20), nullable=True)
    latitude = db.Column(db.Float, nullable=True)
    longitude = db.Column(db.Float, nullable=True)
    este_logat = db.Column(db.Boolean, default=False)
```

Figure 3.27: Definirea modelului Pacient.

Notificări în timp real cu Flask-SocketIO

Un aspect important al aplicației este integrarea notificărilor în timp real. Acest lucru este realizat utilizând *Flask-SocketIO*, care permite trimitera și primirea instantanee de mesaje între server și client.

Exemple de evenimente WebSocket:

```
@socketio.on('connect')
def handle_connect():
    print('Client connected')
    # Fără verificare de token

@socketio.on('disconnect')
def handle_disconnect():
    print('Client disconnected')
```

Figure 3.28: Evenimente pentru conectare și deconectare.

Detalii despre funcționalitate:

- **handle connect:** Această funcție este declanșată atunci când un client se conectează la server. Un mesaj este afișat în terminal pentru a confirma stabilirea conexiunii, facilitând monitorizarea activității în timp real.

- **handle disconnect:** Funcția este apelată la deconectarea unui client. Acest eveniment permite serverului să gestioneze corespunzător starea aplicației și să actualizeze informațiile relevante privind conexiunile active.

Rutele definite în aplicație

Rutele reprezintă punctele de acces prin care utilizatorii interacționează cu aplicația. În **app.py**, rutele sunt definite clar și acoperă funcționalități precum autentificarea utilizatorilor și gestionarea alertelor.

Exemple de rute:

- **/login_dispatcher:** Permite autentificarea dispecerilor. Dacă informațiile introduse sunt corecte, utilizatorul este redirectionat către dashboard-ul dedicat.

```
# Logare dispecerat
@app.route('/login_dispatcher', methods=['GET', 'POST'])
def login_dispatcher():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        dispatcher = Dispatcher.query.filter_by(username=username).first()

        if dispatcher and check_password_hash(dispatcher.password, password):
            session['dispatcher_id'] = dispatcher.id
            return redirect(url_for('dashboard_dispatcher'))
        else:
            flash('Username sau parolă incorectă.')
    return render_template('login_dispatcher.html')
```

Figure 3.29: Codul rutei pentru autentificarea dispecerilor.

- **/dashboard_pacient:** Afisează informații relevante pentru pacienți într-un format accesibil.
- **/dashboard_dispatcher:** Oferă dispecerilor o interfață pentru monitorizarea alertelor și gestionarea intervențiilor.
- **/inregistrare_pacient:** Permite adăugarea unui pacient nou în baza de date utilizând metode POST.

Fisierul **app.py** joacă un rol esențial în buna funcționare a aplicației backend, integrând toate componente cheie ale proiectului. Modelele bine structurate, notificările în timp real și rutele clar definite contribuie la realizarea unei aplicații eficiente, scalabile și ușor de utilizat.

2. Configurarea bazei de date

Baza de date a fost configurată utilizând extensia **SQLAlchemy**, o bibliotecă puternică pentru gestionarea bazelor de date în aplicații Python. Pentru gestionarea modificărilor structurale (migrații), s-a utilizat **Flask-Migrate**. Această combinație asigură atât o interfață intuitivă pentru manipularea datelor, cât și o modalitate ușoară de a actualiza schema bazei de date fără a pierde informații existente.

Inițializarea bazei de date

Inițializarea bazei de date este necesară pentru a pregăti mediul pentru gestionarea migrațiilor. Comanda utilizată pentru acest proces este:

```
flask db init
```

Figure 3.30: Comanda pentru initializarea bazei de date.

Această comandă creează structura necesară în directorul `migrations`, inclusiv fișierele utilizate pentru gestionarea migrațiilor.

Crearea migrațiilor

După definirea modelelor din baza de date, migrațiile sunt generate pentru a reflecta modificările în structura bazei de date. Comanda utilizată pentru a crea prima migrație este:

```
flask db migrate -m "Initial migration"
```

Figure 3.31: Crearea unei migrații initiale pentru baza de date.

Aceasta generează un fișier de migrare care conține detalii despre modificările efectuate.

Aplicarea migrațiilor

Migrațiile create sunt aplicate asupra bazei de date utilizând comanda:

```
flask db upgrade
```

Figure 3.32: Aplicarea migrațiilor asupra bazei de date.

Această comandă sincronizează structura bazei de date cu modelele definite în aplicație.

Modelele tabelelor

Modelele definite în cod includ tabele pentru pacienți, dispeceri și incidente. Fiecare model descrie structura tabelelor și relațiile dintre acestea. În lipsa unor imagini actuale ale tabelelor din aplicație, oferim următoarea reprezentare a structurii tabelelor în format textual:

- **Tabelul Patient:**

- *id*: Identificator unic pentru fiecare pacient.
- *username*, *email*, *password*: Informații de autentificare.
- *prenume*, *nume*, *data_nasterii*: Detalii personale.
- *boala*, *medicamente*: Informații medicale.
- *localizare*: Date despre locația geografică a pacientului.

- **Tabelul Dispatcher:**

- *id*: Identificator unic pentru fiecare dispecer.
- *username*, *password*: Informații de autentificare.

- **Tabelul Accident:**

- *id*: Identificator unic pentru fiecare accident raportat.
- *latitude, longitude*: Coordonatele locației.
- *details*: Detalii despre accident.
- *time*: Data și ora accidentului.

Aceste modele sunt utilizate pentru gestionarea eficientă a datelor din aplicație, iar structura bazei de date poate fi extinsă cu ușurință în funcție de cerințele viitoare.

3. Notificări în timp real cu Flask-SocketIO

Notificările în timp real reprezintă o componentă esențială a aplicației, facilitând comunicarea instantanee între server și utilizatori. Această funcționalitate este implementată utilizând extensia **Flask-SocketIO**, care permite utilizarea protocolului WebSockets pentru o transmisie bidirectională a datelor.

Ce este Flask-SocketIO? Flask-SocketIO este o extensie pentru **Flask** care facilitează implementarea notificărilor în timp real. Aceasta oferă suport pentru mai multe protocoale de transport, inclusiv WebSockets, și permite dezvoltarea de aplicații scalabile și eficiente din punct de vedere al performanței.

Configurarea Flask-SocketIO: Pentru a integra Flask-SocketIO în aplicație, s-au realizat următorii pași:

- **Importarea extensiei:**

```
from flask_socketio import SocketIO, disconnect
```

Figure 3.33: Importarea extensiei Flask-SocketIO.

- **Inițializarea în fișierul principal (app.py):**

```
socketio = SocketIO(app, cors_allowed_origins="*")
```

Figure 3.34: Configurarea Flask-SocketIO în aplicație.

Definirea evenimentelor WebSocket: Evenimentele WebSocket sunt definite pentru a răspunde diferitelor interacțiuni dintre server și clienti. Exemple de evenimente includ:

- **connect:** Gestionează conexiunea unui client la server.

```
@socketio.on('connect')
def handle_connect():
    print('Client connected')
```

Figure 3.35: Evenimentul de conectare al unui client.

- **disconnect:** Gestionează deconectarea unui client.

```
@socketio.on('disconnect')
def handle_disconnect():
    print('Client disconnected')
```

Figure 3.36: Evenimentul de deconectare al unui client.

- **alert:** Gestionează notificările primite de la utilizatori (optional, poate fi definit dacă este nevoie de funcționalitate extinsă pentru alerte).

Executarea aplicației: Aplicația este executată folosind următorul cod pentru a permite notificările în timp real:

```
if __name__ == "__main__":
    socketio.run(app, host='0.0.0.0', port=5000, debug=True)
```

Figure 3.37: Executarea aplicației cu Flask-SocketIO.

Avantajele Flask-SocketIO

- **Comunicație instantanee:** Permite interacțiuni rapide și fără latențe între server și clienți.
- **Scalabilitate:** Suportă mai multe tipuri de protocoale, adaptându-se la nevoile aplicației.
- **Ușor de integrat:** Se conectează eficient cu frontend-ul utilizând clientul Socket.IO.

Implementarea notificărilor în timp real cu Flask-SocketIO îmbunătățește experiența utilizatorilor prin transmiterea rapidă a informațiilor critice. Această funcționalitate joacă un rol esențial în gestionarea alertelor și asigură o interacțiune eficientă în cadrul aplicației.

3.3.4 Funcționalități implementate

3.3.4.1 Fluxul de autentificare

Autentificarea reprezintă primul pas esențial pentru utilizarea aplicatiei, fiind crucială pentru protejarea datelor sensibile și pentru asigurarea accesului controlat la funcționalități. Aceasta este implementată pentru două categorii de utilizatori: **Pacienți** și **Dispeceri**, folosind formulare distincte și o arhitectură bazată pe token-uri *JWT (JSON Web Token)* pentru gestionarea securizată a sesiunilor.

Procesul de autentificare începe printr-o pagină de selecție, unde utilizatorii își aleg rolul (Pacient sau Dispecer). Această abordare modulară permite separarea clară a fluxurilor de lucru pentru fiecare categorie de utilizatori, după cum se observă în Figura 3.38.

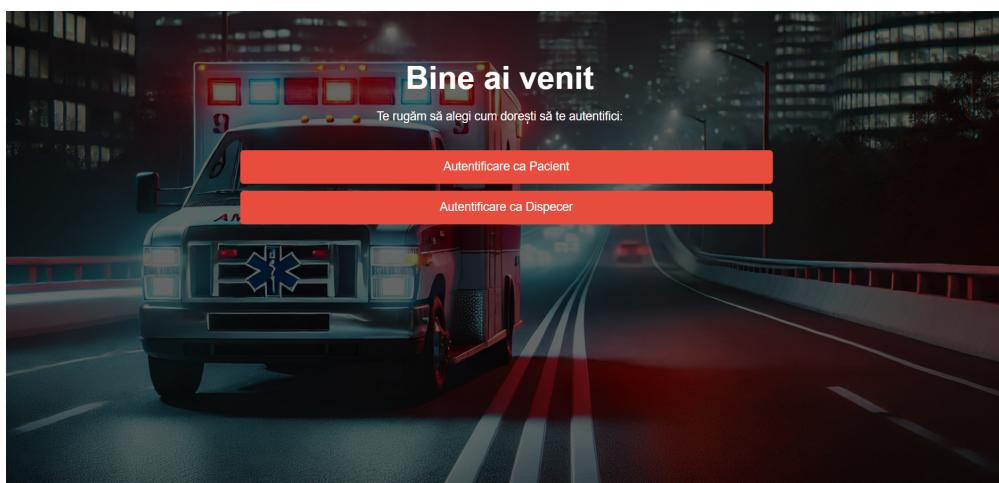


Figure 3.38:

Pagina de selecție pentru autentificare.

1. Autentificarea Dispecerului

Dispecerul beneficiază de un cont unic, generat automat, care îi oferă acces exclusiv la tabloul de bord interactiv pentru gestionarea alertelor. Generarea contului unic presupune utilizarea funcției `generate_password_hash` pentru a crea un hash sigur al parolei. După generare, datele sunt salvate în baza de date, după cum se observă în Figura 3.39.

```
# Datele contului dispeceratului
username = 'Dispecerat112'
password = 'Ambulanta112' # Setează parola dorită

# Generăm hash-ul parolei
hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

# Creăm un nou utilizator de tip Dispatcher
dispatcher = Dispatcher(username=username, password=hashed_password)

# Contextul aplicației Flask
with app.app_context():
    db.session.add(dispatcher)
    db.session.commit()

print('Contul dispeceratului a fost creat cu succes.')
```

Figure 3.39: Codul pentru crearea contului unic al dispecerului.

După generarea contului, dispecerul se poate autentifica utilizând formularul specific rolului său, prezentat în Figura 3.40. Autentificarea dispecerului presupune validarea datelor introduse în formular și verificarea acestora în baza de date.

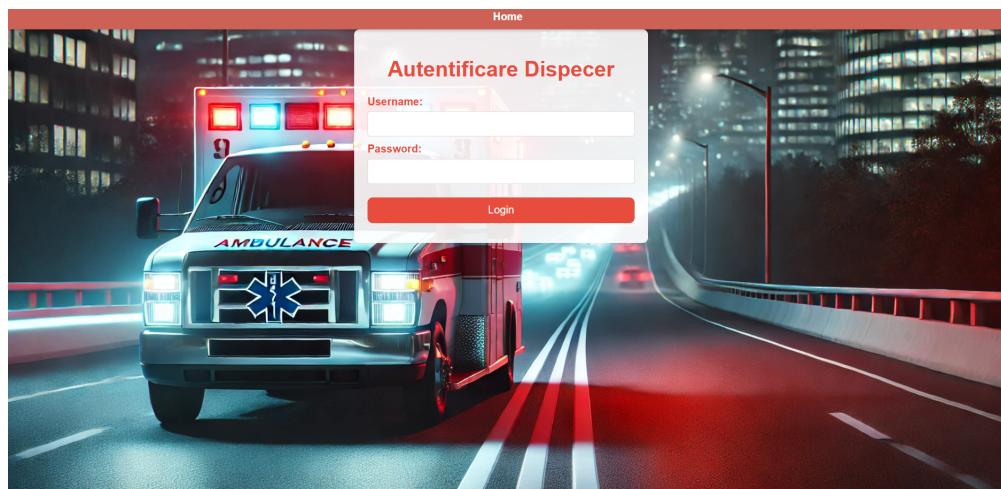


Figure 3.40: Formularul de autentificare pentru dispeceri.

2. Autentificarea Pacientului

Pacienții au posibilitatea de a-și crea un cont înainte de autentificare. Formularul de înregistrare este ilustrat în Figura 3.41. După înregistrare, pacienții pot utiliza formularul de autentificare prezentat în Figura 3.42 pentru a accesa funcționalitățile aplicației.

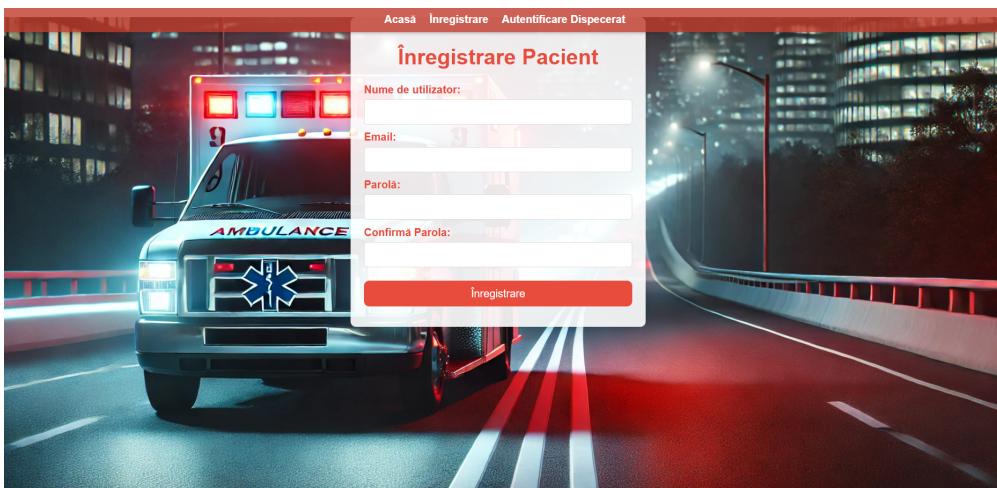


Figure 3.41: *Formularul de înregistrare pentru pacienți.*

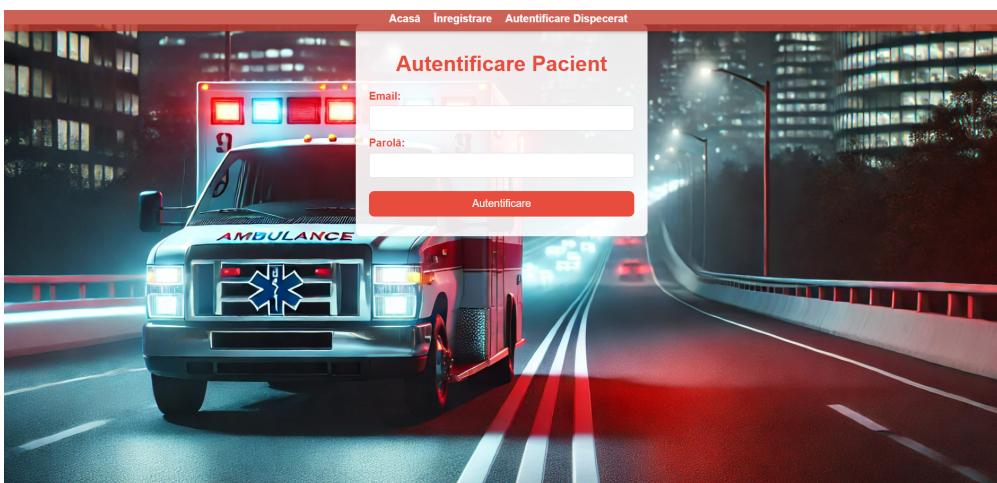


Figure 3.42: *Formularul de autentificare pentru pacienți.*

Autentificarea pacientului presupune validarea email-ului și a parolei furnizate. În cazul unui răspuns valid, pacientul primește acces la dashboard-ul său personalizat.

3. Validarea credibilă a datelor

Validarea datelor utilizatorilor este o etapă esențială în procesul de autentificare, care asigură securitatea și protecția informațiilor sensibile. Aceasta este realizată prin următorii pași:

Verificarea existenței utilizatorului în baza de date. Compararea parolei furnizate cu hash-ul stocat în baza de date, utilizând funcția `check_password_hash`. Generarea unui token JWT pentru utilizatorii autentificați. Codul relevant pentru validarea datelor este ilustrat în Figura 3.43, unde este prezentat întregul proces de autentificare.

```

@app.route('/api/login', methods=['POST'])
def api_login():
    data = request.get_json()

    if not data or 'email' not in data or 'password' not in data:
        return jsonify({"success": False, "message": "Email și parola sunt necesare."}), 400

    email = data['email']
    password = data['password']

    # Găsește pacientul după email
    patient = Patient.query.filter_by(email=email).first()

    if patient and check_password_hash(patient.password, password):
        # Autentificare reușită
        token = generate_token(patient.id) # Generăm un token JWT care conține ID-ul pacientului
        return jsonify({"success": True, "token": token})
    else:
        # Autentificare eșuată
        return jsonify({"success": False, "message": "Nume de utilizator sau parolă incorectă."}), 401

```

Figure 3.43: Codul pentru autentificarea utilizatorilor și validarea datelor.

4. Securizarea sesiunilor

Gestionarea sesiunilor este realizată prin intermediul token-urilor JWT, care conțin informații precum ID-ul utilizatorului. Aceste token-uri expiră după o perioadă prestabilită și sunt verificate la fiecare cerere efectuată de utilizator, pentru a asigura autenticitatea și integritatea sesiunii.

Autentificarea utilizatorilor reprezintă o componentă esențială a aplicatiei, construită pe principii de securitate modernă. Utilizarea hash-urilor pentru parole și token-urilor JWT asigură protecția datelor sensibile, iar implementarea separată a fluxurilor pentru pacienți și dispeceri facilitează o utilizare clară și eficientă.

3.3.4.2 Vizualizare și editare a datelor utilizatorilor

Gestionarea utilizatorilor oferă un mecanism centralizat pentru actualizarea datelor pacienților, sincronizând automat toate modificările între baza de date, platforma web a dispecerului și aplicația mobilă. Această funcționalitate asigură coerenta și disponibilitatea datelor în timp real, indiferent de punctul de acces.

Lista utilizatorilor activi

Dispecerii au acces la o listă actualizată a utilizatorilor activi, care include informații esențiale pentru gestionarea alertelor și a urgențelor. Această listă, ilustrată în Figura 3.44, conține:

- **Informații personale:** Nume, prenume, email.
- **Starea medicală a pacientului:** Boala și medicamentele curente.
- **Locația pacientului:** Oraș, adresă.
- **Informații de contact:** Număr de telefon.

Modificările efectuate de pacient sau dispecer sunt reflectate instantaneu în tabelul afișat pe platforma web, după reîncărcarea paginii.

		Dashboard		Pacienți Logați	Deconectare		
Nume	Prenume	Email	Boala	Medicamente		Oras	Nr. Telefon
Anda	Iacinschi	anda.roxana.iacinschi@gmail.com	Scleroză laterală amiotrofică (SLA)	Riluzol, Edaravone		Dorohoi	0751537895
Ioan	Popescu	ion.popescu@gmail.com	Paralizie cerebrală	Baclofen, Gabapentin		Cluj-Napoca	0741234567
Maria	Ionescu	amaria.ionescu@gmail.com	Scleroză multiplă	Interferon beta-1a, Methylprednisolone		București	0751234567
Mihai	Grigorescu	mihai.grigorescu@gmail.com	Accident vascular cerebral (hemiplegie)	Aspirin, Clopidogrel		Brasov	4078123456
Ana	Stoian	ana.stoian@gmail.com	Paralizie spastică	Tizanidina, Diazepam		Constanta	0791234567
None	None	gabriela.marin@gmail.com	Scleroză laterală amiotrofică (SLA)	Riluzol, Edaravone		Voluntari	0701345678
Florin	Vasilescu	florin.vasilescu@gmail.com	Atrofie musculară spinală	Nusinersen, Risdiplam		Oradea	0711234567
Camelia	Voicu	camelia.voicu@gmail.com	Miastenia gravis	Pyridostigmine, Azathioprine		Galati	0721234567
Radu	Minculescu	radu.minculescu@gmail.com	Leziune traumatică a măduvei spinări (paraplegie)	Baclofen, Dantrolen		Pitești	0731234567

© Asistent Mobil

Figure 3.44: Tabelul utilizatorilor activi vizibil dispecerului.

Actualizarea fișei pacientului

Fişa pacientului permite acestuia să-şi editeze informaţiile personale printr-un formular complet, prezentat în Figura 3.45. Formularul include câmpuri pentru:

- Informații personale:** Prenume, nume, data nașterii.
- Informații medicale:** Boala, medicamentele curente.
- Adresă:** Țară, județ, oraș, stradă, număr, cod poștal.
- Contact:** Număr de telefon.
- Ultimul incident:** Detalii relevante pentru istoricul medical.

Figure 3.45: Formularul de actualizare a fișei pacientului.

Procesul de actualizare și sincronizare automată

Când pacientul își actualizează fișa, procesul de sincronizare este gestionat automat, urmând pașii următori:

1. Trimiterea datelor la server:

- Modificările introduse în formularul pacientului sunt trimise către server printr-o cerere API.
- Serverul validează datele primite (ex. verifică formatul câmpurilor, asigură integritatea datelor).

2. Actualizarea bazei de date:

- Serverul actualizează informațiile pacientului în baza de date principală.
- În cazul modificării adresei, aceasta este procesată printr-o funcție de geocodificare pentru a obține coordonatele GPS (latitudine și longitudine).

3. Sincronizarea pe platforma web a dispecerului:

- După ce datele sunt actualizate în baza de date, serverul trimite o notificare prin WebSocket către platforma web.
- Platforma web a dispecerului actualizează automat tabelul utilizatorilor activi, afișând cele mai recente informații.

4. Sincronizarea pe aplicația mobilă:

- Notificarea trimisă prin WebSocket este transmisă și către aplicația mobilă, care actualizează fișa pacientului în timp real.
- Această sincronizare asigură coerența între platforma mobilă și cea web.

Beneficii ale sincronizării automate

- **Actualizare în timp real:** Toate modificările sunt propagate instantaneu între platformele conectate.
- **Reducerea erorilor:** Datele sunt centralizate și validate, eliminând riscul de inconsistență între aplicații.
- **Eficiență operațională:** Dispecerii au întotdeauna acces la cele mai recente informații despre pacienți, esențiale în gestionarea situațiilor de urgență.

```

# Rută pentru actualizarea profilului pacientului
@app.route('/update_profile', methods=['GET', 'POST'])
def update_profile():
    if 'patient_id' not in session:
        return redirect(url_for('login_patient'))

    pacient = Patient.query.get(session['patient_id'])

    if request.method == 'POST':
        pacient.prenume = request.form['prenume']
        pacient.nume = request.form['nume']
        pacient.data_nasterii = datetime.strptime(request.form['data_nasterii'], '%Y-%m-%d').date()
        pacient.boala = request.form['boala']
        pacient.medicamente = request.form['medicamente']
        pacient.ulimul_incident = request.form['ulimul_incident']
        pacient.tara = request.form[' Tara ']
        pacient.judet = request.form['judet']
        pacient.oras = request.form['oras']
        pacient.strada = request.form['strada']
        pacient.numar = request.form['numar']
        pacient.cod_postal = request.form['cod_postal']
        pacient.telefon = request.form['telefon']

        # Geocodifică adresa și salvează coordonatele în baza de date
        lat, lon = geocode_address(pacient.tara, pacient.judet, pacient.oras, pacient.strada, pacient.numar, pacient.cod_postal)
        print(f"Coordonatele obținute: latitudine={lat}, longitudine={lon} " # Debugging
        if lat and lon:
            pacient.latitude = lat
            pacient.longitude = lon
        else:
            flash("Nu s-au putut obține coordonatele pentru această adresă.", 'danger')

        # Verificăm dacă latitudinea și longitudinea au fost setate corect
        if pacient.latitude and pacient.longitude:
            print(f"Latitudinea și longitudinea au fost setate corect: ({pacient.latitude}, {pacient.longitude})")
        else:
            print("Latitudinea și longitudinea nu au fost setate corect.")

        db.session.commit()
        flash('Profil actualizat cu succes!', 'success')
        return redirect(url_for('dashboard_patient'))

    return render_template('update_profile.html', pacient=pacient)

```

Figure 3.46: Codul pentru actualizarea profilului pacientului.

Funcționalitatea de gestionare a utilizatorilor facilitează actualizarea datelor pacienților și sincronizarea acestora între platformele conectate. Sincronizarea automată asigură accesul la informații actualizate, esențiale pentru eficiență și siguranță operațională.

3.3.4.3 Gestiona alertelor și vizualizarea pe hartă

Gestiona alertelor și vizualizarea pacienților pe hartă reprezintă o funcționalitate esențială a platformei web, oferind dispecerilor o imagine clară și actualizată asupra locațiilor și stării pacienților. Această secțiune detaliază modul în care sunt afișați pacienții autentificați și cum sunt gestionate notificările de urgență.

Vizualizarea pacienților pe hartă

Platforma web oferă dispecerilor o hartă interactivă care afișează în timp real locațiile pacienților autentificați. Această funcționalitate asigură o monitorizare eficientă și permite o intervenție rapidă în caz de urgență. Harta utilizează coordonatele GPS stocate în baza de date și este integrată cu sistemul de gestionare a utilizatorilor. Pentru implementarea acestei funcționalități, au fost utilizate tehnologii moderne, precum:

- **Leaflet.js** – utilizată pentru afișarea hărții și plasarea markerilor, oferind o interfață interactivă și intuitivă pentru dispeceri.
- **Integrarea cu baza de date prin API-ul backend** – coordonatele GPS și informațiile relevante despre pacienți sunt preluate din baza de date și afișate pe hartă în timp real.

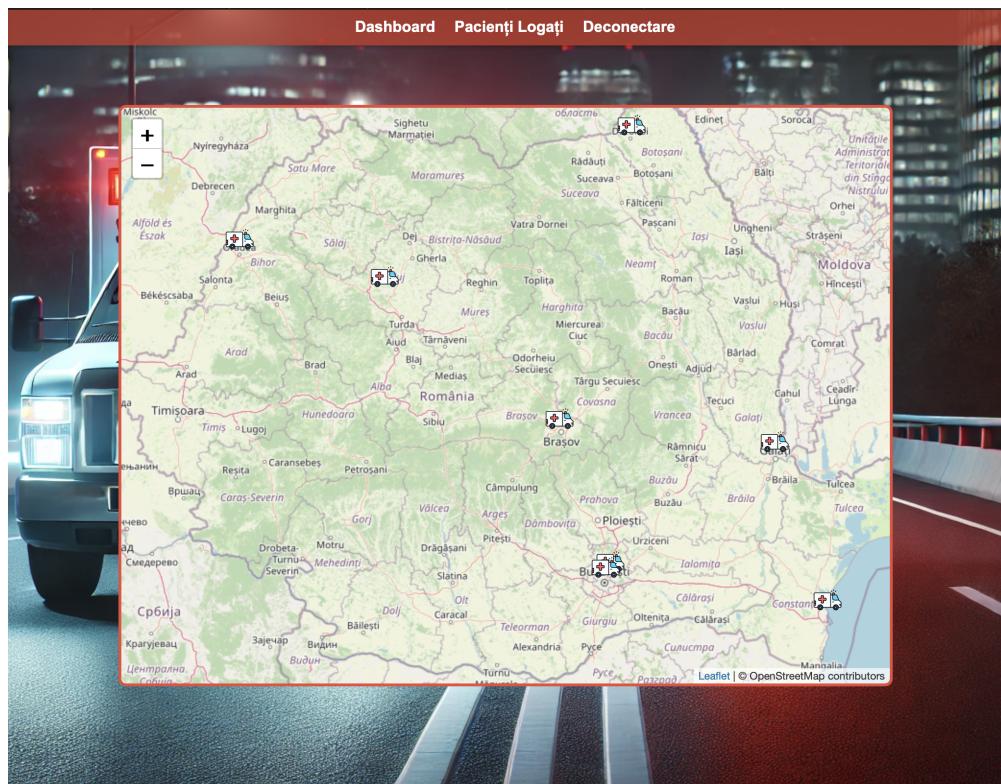


Figure 3.47: Harta cu locațiile pacienților autentificați.

Markerii de pe hartă

Fiecare pacient este reprezentat printr-un marker personalizat, afișat pe baza coordonatelor GPS ale acestuia. Markerii sunt vizibili clar pe hartă, oferind o reprezentare precisă a poziției fiecărui pacient. Acest sistem permite dispecerilor să identifice rapid locațiile pacienților.

Informații afișate la mouse-over

Când dispecerul trece cu mouse-ul peste un marker, sunt afișate următoarele informații relevante:

- **Numele și prenumele pacientului** – pentru identificare rapidă.
- **Boala cronică și medicația curentă** – oferind informații medicale esențiale pentru o intervenție bine informată.
- **Numărul de telefon** – pentru contact direct și rapid cu pacientul sau cu persoanele responsabile de acesta.

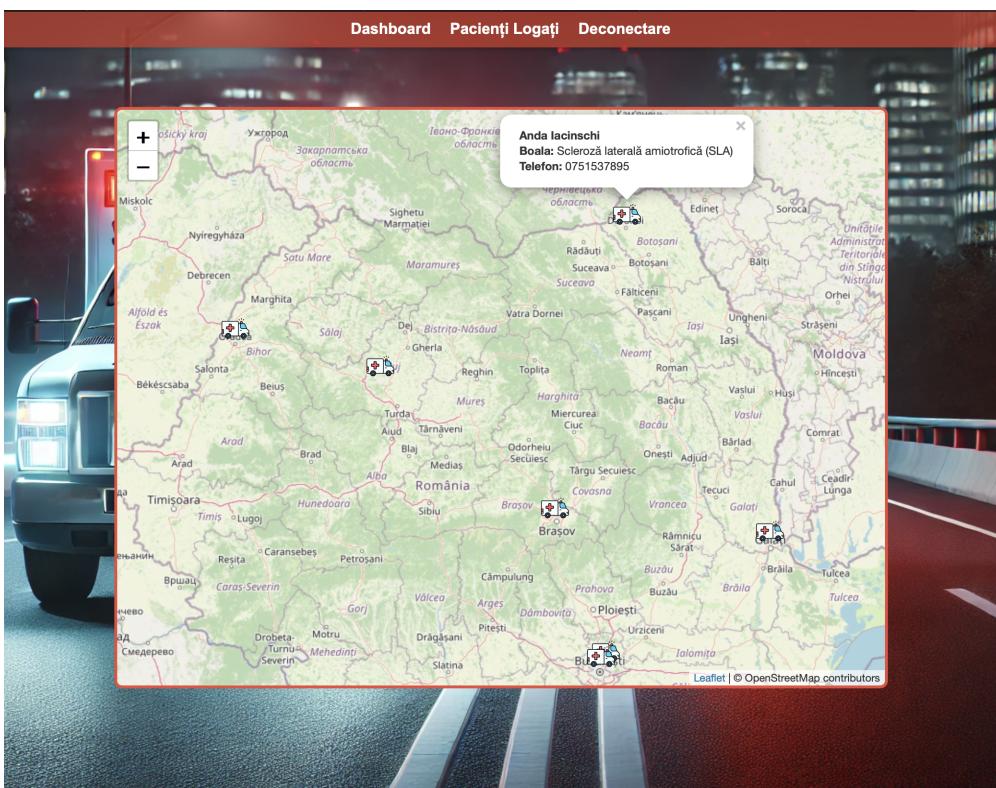


Figure 3.48: *Informații afișate la mouse-over pentru un pacient.*

Integrarea cu baza de date și actualizările în timp real

Harta este conectată direct la baza de date a aplicației, ceea ce asigură actualizarea automată a locațiilor pacienților. În momentul în care un pacient se autentifică și își actualizează coordonatele GPS, markerul de pe hartă este actualizat automat, fără intervenție suplimentară din partea dispecerului.

Această sincronizare în timp real este realizată folosind WebSocket, care transmite modificările de la server către interfața web în mod continuu, oferind o experiență fluidă și fără întreruperi pentru utilizatori.

Vizualizarea pacienților pe hartă reprezintă o funcționalitate critică a platformei, care îmbunătățește semnificativ capacitatea dispecerilor de a monitoriza și răspunde eficient la urgențe. Integrarea datelor GPS și sincronizarea în timp real oferă o soluție modernă și fiabilă pentru gestionarea situațiilor critice.

Notificări de impact detectat

Platforma este configurată să alerteze dispecerii în cazul în care un pacient raportează un impact sau o urgență. Această funcționalitate este crucială pentru o reacție rapidă și bine informată. Notificările includ detalii esențiale despre locația accidentului și starea pacientului, oferind dispecerilor informațiile necesare pentru a coordona o intervenție.

Detectarea automată a impactului

Aplicația mobilă detectează automat un impact prin intermediul senzorilor dispozitivului (cum ar fi accelerometrul și giroscopul). În momentul detectării unui eveniment, aplicația generează o notificare care este transmisă către server.

- **Procesarea notificării pe server:** Serverul primește datele, care includ locația pacientului, ora impactului și detalii despre starea acestuia. Acestea sunt prelucrate și transmise către platforma web a dispecerului.

Informațiile incluse în notificare Notificarea oferă dispecerilor o imagine completă asupra situației, incluzând:

- **Numele, boala și istoricul medical al pacientului:** Ajută la înțelegerea riscurilor implicate.
- **Coordonatele GPS ale locației impactului:** Sunt afișate pe hartă pentru a facilita identificarea rapidă a locului.
- **Starea pacientului:** Detaliată într-un mesaj care poate indica, de exemplu, că pacientul nu răspunde, necesită o ambulanță sau are o situație critică.

Fluxul notificării

- Aplicația mobilă detectează evenimentul și transmite notificarea către server.
- Serverul procesează datele și notifică în timp real dispeceratul prin WebSocket.
- Harta interactivă afișează markerul pacientului cu un simbol de alertă (de exemplu, marker roșu).
- Dispecerul primește notificarea detaliată și poate acționa imediat.

Scenarii particulare Notificările sunt personalizate în funcție de starea pacientului:

- **Pacientul se simte bine:** Notificarea precizează că pacientul este în afara oricărui pericol, iar markerul rămâne vizibil pe hartă.

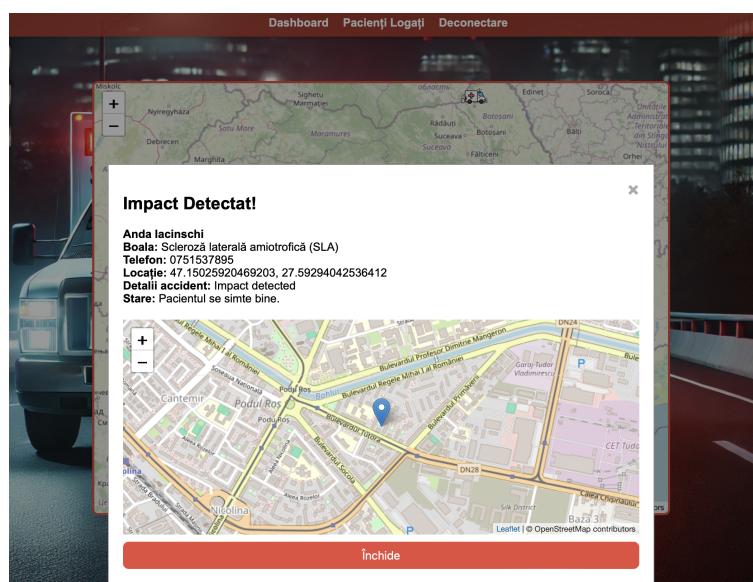


Figure 3.49: Notificare de impact detectat pentru un pacient care se simte bine.

- **Pacientul nu se simte bine:** Notificarea subliniază necesitatea unei intervenții urgente, cum ar fi trimiterea unei ambulanțe.

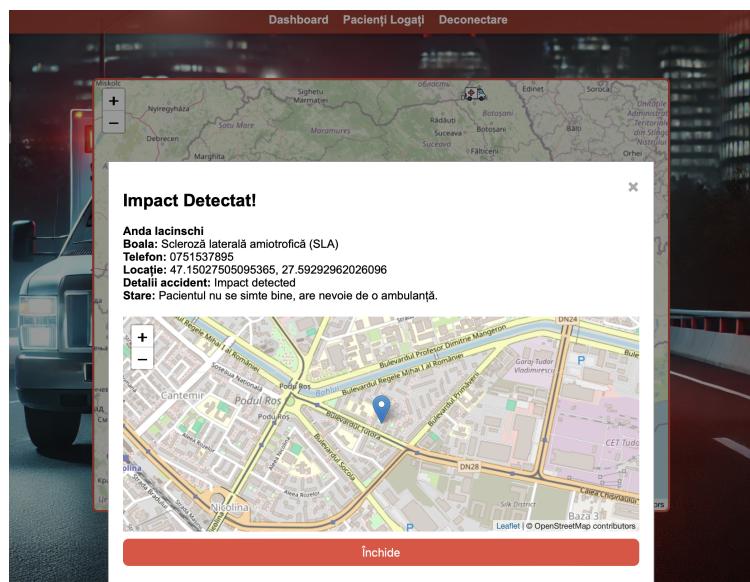


Figure 3.50: Notificare de impact detectat pentru un pacient care necesită intervenție.

- Pacientul nu răspunde:** Sistemul marchează situația ca fiind critică, iar dispecerii sunt încurajați să ia măsuri suplimentare, cum ar fi contactarea directă sau accesarea informațiilor detaliate ale pacientului.

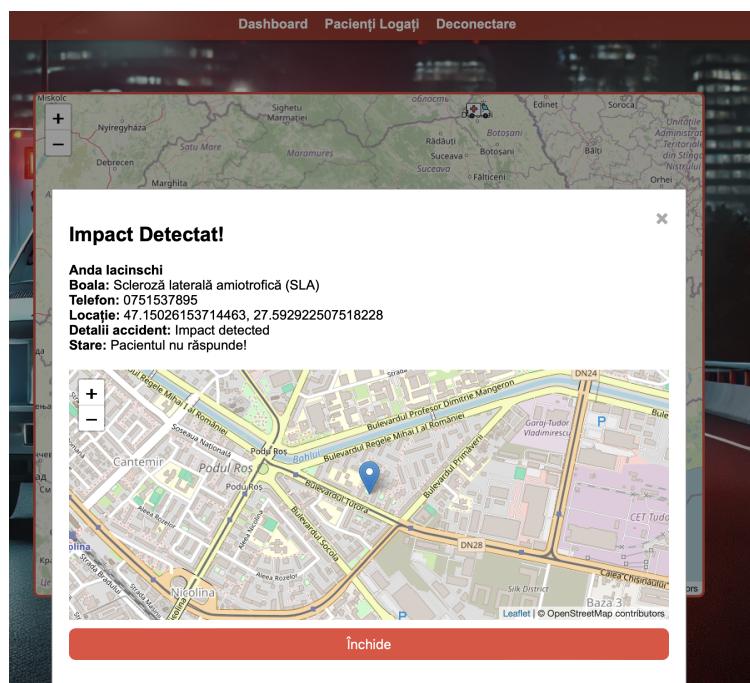


Figure 3.51: Notificare de impact detectat pentru un pacient care nu răspunde.

Această funcționalitate permite dispecerilor să gestioneze eficient urgențele și să monitorizeze locațiile pacienților în timp real. Notificările detaliate și integrarea hărții cu baza de date asigură o reacție rapidă și informată în situații critice.

Capitolul 4 Concluzii si implementari viitoare

4.1 Concluzii

Lucrarea de față a avut ca obiectiv dezvoltarea unui sistem integrat format dintr-o aplicație mobilă și o platformă web, care să sprijine persoanele cu mobilitate redusă. Prin acest proiect, s-au demonstrat atât beneficiile tehnologiilor moderne, cât și capacitatea lor de a rezolva probleme reale, cu un impact semnificativ asupra utilizatorilor. Iată concluziile detaliate referitoare la aplicația mobilă și platforma web:

Aplicația mobilă

1. Detectarea automată a căderilor: Aplicația mobilă, dezvoltată nativ pentru platforma iOS folosind Swift, utilizează accelerometru integrat în telefon pentru a detecta căderile. Algoritmii proiectați au fost optimizați pentru a reduce alertele false și pentru a asigura un timp de răspuns minim.
2. Interfață intuitivă: Designul aplicației mobile este simplu și accesibil, ceea ce permite utilizatorilor, inclusiv persoanelor cu un nivel scăzut de alfabetizare tehnologică, să o folosească eficient. Funcționalitățile cheie, precum alertele și accesarea profilului utilizatorului, sunt disponibile direct din ecranul principal.
3. Notificările în caz de urgență: În cazul unei căderi detectate, aplicația declanșează automat notificări pentru a verifica starea utilizatorului. Dacă acesta nu răspunde în timp util, se trimit o alertă către platforma web a dispeceratului medical.
4. Configurabilitatea: Aplicația permite ajustarea pragurilor de sensibilitate ale senzorilor, astfel încât să se potrivească nevoilor specifice ale utilizatorului.
5. Monitorizarea continuă: Aplicația rulează în fundal fără a afecta semnificativ performanța dispozitivului sau consumul de baterie, ceea ce o face ideală pentru utilizare zilnică.

Platforma web

1. Monitorizarea în timp real: Platforma web a fost proiectată pentru dispeceri medicali și permite vizualizarea în timp real a alertelor transmise de aplicația mobilă. Datele includ informații despre locația utilizatorului, istoricul incidentelor și detalii personale relevante.
2. Interfață pentru dispeceri: Tabloul de bord centralizat afișează toate alertele active și istoricul acestora, facilitând prioritizarea intervențiilor. Interfața este gândită pentru a fi ușor de navigat, chiar și în situații de stres.

3. Securitatea datelor: Toate datele transmise între aplicația mobilă și platformă sunt criptate, iar accesul la platformă este protejat prin autentificare cu token-uri JWT. Aceasta garantează respectarea normelor GDPR și protecția informațiilor personale ale utilizatorilor.
4. Gestionarea utilizatorilor: Platforma permite dispecerilor să acceseze și să actualizeze profilurile utilizatorilor, inclusiv date medicale, contacte de urgență și alte informații utile pentru coordonarea intervențiilor.
5. Integrarea cu hărți interactive: Locația utilizatorilor este afișată pe o hartă interactivă, ceea ce permite o localizare rapidă și precisă. Aceasta ajută dispecerii să ia decizii informate și să coordoneze echipele de intervenție.

Rezultate generale

Proiectul a reușit să integreze eficient cele două componente, oferind un sistem robust și scalabil. Aplicația mobilă și platforma web colaborează în timp real, asigurând o experiență fluidă și eficientă pentru utilizatori și personalul medical. Acest sistem nu doar că îmbunătățește siguranța utilizatorilor, ci și demonstrează cum soluțiile tehnologice pot avea un impact semnificativ asupra calității vietii.

4.2 Posibile Dezvoltări

1. Extinderea compatibilității platformei

Un pas esențial este dezvoltarea unei versiuni a aplicației mobile pentru platforma Android. Acest lucru ar crește accesibilitatea sistemului pentru o gamă mai largă de utilizatori și ar asigura interoperabilitatea între diverse dispozitive. De asemenea, integrarea cu platforme wearable (ex. ceasuri inteligente) ar oferi monitorizare suplimentară și mai detaliată.

2. Detectarea avansată a incidentelor

Pentru a crește eficiența sistemului, algoritmii pot fi extinși pentru a detecta alte tipuri de incidente, cum ar fi convulsiile sau episoadele de pierdere a conștiinței. Utilizarea algoritmilor de învățare automată ar permite analiza datelor în timp real și ar reduce semnificativ rata alertelor false.

3. Funcționalități de comunicare îmbunătățite

Adăugarea unui sistem de chat vocal sau video între utilizator și dispecerat ar facilita evaluarea rapidă a situației utilizatorului. De asemenea, funcționalitatea de răspuns automat al dispeceratului ar permite gestionarea eficientă a cazurilor mai puțin urgente.

4. Integrarea cu dosare electronice de sănătate (EHR)

Un alt pas important este conectarea platformei web cu sisteme medicale existente, cum ar fi dosarele electronice de sănătate. Această integrare ar oferi personalului medical acces la istoricul complet al pacientului, îmbunătățind deciziile de intervenție.

5. Monitorizarea și analiza datelor pe termen lung

Implementarea unui modul analitic care să colecteze și să analizeze datele pe termen lung ar permite generarea unor rapoarte personalizate pentru utilizatori și medici. Aceste rapoarte ar putea include tendințe privind activitatea fizică, frecvența incidentelor și starea generală de sănătate a utilizatorilor.

Prin implementarea acestor direcții de dezvoltare, aplicația poate deveni un instrument indispensabil pentru persoanele vulnerabile, contribuind semnificativ la creșterea siguranței și a calității vietii acestora.

Bibliografie

- [1] World Health Organization. *World Report on Disability*. 2011. URL: <https://www.who.int/publications-detail-redirect/9789241564182>.
- [2] Eurostat. “Disability Statistics Explained”. In: (2022). URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Disability_statistics.
- [3] Consiliul Național pentru Combaterea Discriminării. *Raport privind accesibilitatea instituțiilor publice din România*. 2021. URL: <https://www.cncd.ro/wp-content/uploads/2022/04/Raport-de-activitate-CNCD-2021-.pdf>.
- [4] World Health Organization. *Global Strategy on Digital Health 2020–2025*. 2021. URL: <https://www.who.int/publications-detail/digital-health-strategy>.
- [5] Teladoc Health. *Telemedicine Services and Benefits*. 2022. URL: <https://www.teladochealth.com>.
- [6] Reflectio Blog. *Impactul inteligenței artificiale asupra sănătății: Progrese în diagnosticare, tratament și prevenirea bolilor*. 2023. URL: <https://reflectio.ro/impactul-inteligentei-artificiale-asupra-sanatatii-progrese-in-diagnosticare-tratament-si-prevenirea-bolilor/>.
- [7] World Health Organization. *Falls: Key Facts*. 2018. URL: <https://www.who.int/news-room/fact-sheets/detail/falls>.
- [8] National Institute on Aging. *Social Isolation and Loneliness in Older Adults: Opportunities for the Health Care System*. 2021. URL: <https://www.nia.nih.gov/news/social-isolation-loneliness-older-adults>.
- [9] A. Smith and B. Johnson. “Wearable Technology for Fall Detection in Older Adults: A Review”. In: *Journal of Smart Health Technologies* 10 (2022), pp. 15–30. URL: <https://example.com/fall-detection-wearable-2022>.

- [10] Life Alert Inc. “Life Alert: Emergency Response for Seniors”. In: (2020). URL: <https://www.lifealert.com>.
- [11] Medtronic Inc. “Guardian Connect: Continuous Glucose Monitoring System”. In: (2022). URL: <https://www.medtronicdiabetes.com/products/guardian-connect-continuous-glucose-monitoring-system>.
- [12] Apple Inc. *Health Features of Apple Watch*. 2023. URL: <https://www.apple.com/apple-watch-series-8/health/>.
- [13] Fitbit Inc. *Fitbit Devices: Health and Wellness Monitoring*. 2023. URL: <https://www.fitbit.com/global/us/technology>.
- [14] GeeksforGeeks. *Various Smartphone Sensors*. 2023. URL: <https://www.geeksforgeeks.org/various-smartphone-sensors/>.
- [15] Apple Inc. *Getting Raw Accelerometer Events*. 2023. URL: <https://developer.apple.com/documentation/coremotion/getting-raw-accelerometer-events>.
- [16] Gadget Review. *Wi-Fi 6 și 5G: Cum îmbunătățesc conectivitatea în aplicațiile mobile*. 2023. URL: <https://gsc-online.ro/utilizarea-wi-fi-6-si-5g-pentru-construirea-infrastructurilor-avansate/>.
- [17] Tech Insights. *5G – O revoluție în telecomunicații și conectivitate*. 2023. URL: <https://androit.tech/5g/>.
- [18] Lebara Blog. *Un ghid aprofundat al performanțelor CPU în procesoarele pentru smartphone-uri*. 2023. URL: <https://blog.lebara.co.uk/un-ghid-aprofundat-al-performantelor-cpu-in-procesoarele-pentru-smartphone-uri>.
- [19] ARM Developer. *CPU Architectures for Mobile Devices*. Accesat pe 20 ianuarie 2025. 2023. URL: <https://developer.arm.com/solutions/mobile>.
- [20] Mobile Direct Blog. *Evoluția smartphone-urilor cu AI*. 2023. URL: <https://blog.mobiledirect.ro/evolutia-smartphone-urilor-cu-ai>.
- [21] DRMedia. *Evoluția smartphone-urilor: de la telefon la supercomputer portabil*. 2023. URL: <https://drmedia.ro/evolutia-smartphone-urilor-de-la-telefon-la-supercomputer-portabil>.