**CodeWorks**

**~ SISTEM PURTABIL DE SUPRAVEGHERE A STĂRII DE SĂNĂTATE ~**

**O imagine care conține antenă, obiect

Descrierea a fost generată cu un grad foarte mare de încredere**



**Proiectare**

**Arhitectura programului**

1. Tehnologii si medii de dezvoltare

Proiectul Awear se bazează pe mai multe tehnologii si medii de dezvolatare pentru fiecare modul in parte:

* Modulul SA:

Mediul de dezvoltare: Arduino IDE, care este o aplicație cross-platform scrisă în java și suportă limbajele de programare C si C++ folosind reguli spciale de organizare a codului pentru a avea execuție corectă pe suportul hardware

Tehnologii: C/C++ cu library embedded (ex. Wiring)

* Modulul APS:

Mediul de dezvoltare: Android Studio IDE, environment-ul oficial de la Google pentru sistemul lor de operare Android.

Tehnologii: Java

* Modulul CLOUD:

Mediul de dezvoltare: Microsoft Visual Studio, ce include un set complet pentru dezvoltarea unui backend cu ajutorul serviciilor oferite de Azure

Tehnologii: C#

* Modulul Web:

Mediul de dezvoltare: Microsoft Visual Code, editor de cod sursa ce include support pentru InteliSense, Git, Snippets si multe alte plug-in-uri pentru majoritatea limbajelor de programare.

Tehnologii: HTML5, CSS3, JavaScript

1. Client-server

Sistemul celor 4 module comunică între ele sub forma a mai multor relații client server.

|  |  |  |
| --- | --- | --- |
| Client | Server | Descriere |
| APS | SA | Trimiterea datelor de la sistemul de achizitie la Aplicatia Pacient- SMARTPHONE |
| Cloud | APS | Intre Cloud si Aplicatia Pacient- SMARTPHONE exista o relatie bidirectionala |
| APS | Cloud | Ambele ajungand si server si client |
| WEB | Cloud | Trimitera datelor de la Cloud la modulul Web |

3. Modul in care funcționează fiecare modul

Sistemul AWear este împărțit pe 4 module care sunt strâns legate.

* Sistemul de achiziție (SA)

SA se bazează pe o componentă hardware care, cu ajutorul unor senzori, adună date medicale în timp real și are ca scop principal trimiterea acestora la un smartphone personal.

* Aplicație pacient smartphone (APS)

APS reprezintă aplicația ce preia datele de la sistemul de achiziție pe care apoi le procesează pentru a le afișa în cadrul aplicației si pentru a le trimite mai departe în cadrul unui cont personal care ține evidenta acestora pe un server cloud. De asemenea, are scopul de a face avertizări și a afisa recomandări date de către un medic ce se ocupă de starea pacientului asociat aplicației.

* Sistemul Cloud

Cloud reprezintă serverul ce ține o bază de date cu toate informațiile salvate în urma primirii acestora de la aplicația smartphone. De asemenea, găsim datele de logare și personale ale pacienților și medicilor ce folosesc sistemul AWear.

* Aplicația Web

Pe partea de Web avem interfața importantă pentru medici de unde aceștia monitorizează pacienții și trimit avertizări sau recomandări prin intermediul serviciului de cloud inapoi la aplicația smartphone. Fiecare pacient poate să vadă un istoric cu starea lor primită de la senzori cât și istoricul avertizărilor / recomandărilor

4. Diagrama clase Aplicație Pacient-SMARTPHONE

**O imagine care conține text, captură de ecran

Descrierea a fost generată cu un grad foarte mare de încredere**

4.2 Backend

**O imagine care conține captură de ecran

Descrierea a fost generată cu un grad foarte mare de încredere**

5. Diagrama de stare

O imagine care conține text, hartă

Descrierea a fost generată cu un grad foarte mare de încredere

**Descrierea componentelor(modulelor)**

**1. Modulul Sistem de achiziție (SA)**

Componenta principală a sistemului de achiziție este reprezentată de o structură:

SensorData

* tempData - float (temperatura în grade celsius)
* pulseData - int (valoarea pulsului în bătăi pe minute)
* humData - int (procentul umidității)
* ECGData – int [2] (valoarea sistolei și a diastolei)

Alte componente:

* citirea datelor: readTemp, readPulse, readHum, readECG; folosite pentru citirea datelor de la senzori
* actualizarea datelor din structura principala(SensorData): updateStatistics(sensorData)
* trasmiterea datelor: sendData(sensorData)

**2. Modulul Aplicație Pacient-SMARTPHONE (APS)**

Componentele aplicației smartphone sunt împărțite pe activități:

* + - LogInActivity
    - AccountActivity
    - RecommandationActivity
    - MonitoringActivity
    - ProfileActivity

Fiecare dintre acestea au urmatoarele componente:

* + - Connectivity - pentru verificarea conecțiunii la Internet si Bluetooth
    - CloudData - pentru a transmite sau/și primi date de la Cloud
    - DeviceData - pentru a transmite sau/și primi date de la SA
    - Account - pentru a reține datele de cont introduse de utilizator
    - Recomandation - pentru a salva datele pe care le primește utilizatorul din cloud referitoare la recomandările medicului
    - Monitoring - pentru a salva datele care vin in timp real de la SA
    - Profile - pentru a salva datele de profil personal ale utilizatorului

**3. Modulul Aplicație Cloud**

Componentele modului cloud se leagă de căile de acces la date:

* **/Api/Login** - calea unde se verifică datele de logare ale unui utilizator
* **/Api/Pacients/{id}/Warnings** - calea de unde se iau informații legate de advertismentele pacienților
* **/Api/Pacients/{id}/Recomandations** - calea de unde se iau informații legate de recomandările pacienților
* **/Api/Pacients/{id}/SensorData** - calea de unde se iau informații legate de datele primite de la senzorii (SA) pacienților
* **/Api/Pacients/{id}/PacientFile** - calea de unde se iau informații legate de fișa medicală a pacienților
* **/Api/Medics/{id}/PacientsList** - calea de unde se iau pacienții asociati unui medic
* **/Api/Medics/{id}/PacientsList/{PacientId}** - calea spre un pacient anume asociat unui medic

**4. Modulul Aplicație Web**

4.1 Căile de acces la datele din cloud (Fetchuri):

* Fetch-ul folosit pentru datele de logare ale unui pacient sau ale unui medic: **fetch('/API/Login')**
* Fetch-ul folosit pentru datele de warning din baza de date:

**fetch('API/Pacients/ID/Warnings')**

* Fetch-ul folosit pentru datele de recomandare din baza de date:

**fetch('API/Pacients/ID/Recommandations')**

* Fetch-ul folosit pentru datele primite de la senzori din baza de date:

**fetch('API/Pacients/ID/SensorsData')**

* Fetch-ul folosit pentru datele pacientului din baza de date:

**fetch('API/Pacients/ID/PacientFile')**

* Fetch-ul folosit pentru datele din lista pacienților din baza de date:

**fetch('API/Medics/ID/PacientsList')**

* Fetch-ul folosit pentru datele din lista pacienților din baza de date în funcție de ID-ul acestora:

**fetch('API/Medics/ID/PacientsList/PacientID')**

4.2 Clasele si variabilele din JavaScript ce stochează informația și **creează** paginile web:

* loginInfo
* warningData
* recommandationsData
* sensorData

Acestea conțin date pe care le dețin atât pacienții cât și medicii despre pacienții lor.

**Descrierea comunicării între module**

**Diagrame de secvență**

**O imagine care conține hartă, text

Descrierea a fost generată cu un grad foarte mare de încredere**

**O imagine care conține text, hartă

Descrierea a fost generată cu un grad foarte mare de încredere**

**O imagine care conține text, hartă

Descrierea a fost generată cu un grad foarte mare de încredere**

1. **Modulul Sisteme de Achiziție (SA)**

Având drept dispozitiv un Lilypad, vom folosi **biblioteca SoftwareSerial** pentru a stabili o conexiune serială de tip UART intre Lilypad Arduino si Android:

**#include "SoftwareSerial.h";**

Biblioteca SoftwareSerial a fost dezvoltată pentru a permite communicatie serială pe alți pini digitali Arduino (pe langa comunicație nativă de tip hardware pe pini consacrati PIN0 si PIN1).

Aceasta se foloseste de software pentru a replica această funcționalitate de UART.

**SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);**

Chip-ul Bluetooth are rata de transfer configurată la 115 200; pentru a-l folosi la un baud rate mai scăzut (9600) se folosește un cod ce utilizează o conexiune de tip Software Serial între Arduino si Bluetooth care îl configurează pentru sesiunea curentă înainte de a fi utilizat.

Funcții:

* [SoftwareSerial](https://www.arduino.cc/en/Reference/SoftwareSerialConstructor)()
* [available](https://www.arduino.cc/en/Reference/SoftwareSerialAvailable)()
* [begin](https://www.arduino.cc/en/Reference/SoftwareSerialBegin)()
* [isListening](https://www.arduino.cc/en/Reference/SoftwareSerialIsListening)()
* [overflow](https://www.arduino.cc/en/Reference/SoftwareSerialOverflow)()
* [peek](https://www.arduino.cc/en/Reference/SoftwareSerialPeek)()
* [read](https://www.arduino.cc/en/Reference/SoftwareSerialRead)()
* [print](https://www.arduino.cc/en/Reference/SoftwareSerialPrint)()
* [println](https://www.arduino.cc/en/Reference/SoftwareSerialPrintln)()
* [listen](https://www.arduino.cc/en/Reference/SoftwareSerialListen)()
* [write](https://www.arduino.cc/en/Reference/SoftwareSerialWrite)()

**Conexiuni Hardware**

* Bluetooth CTS-I - nu se conectează
* Bluetooth VCC - Arduino 5V
* Bluetooth GND - Arduino GND
* Bluetooth TX-O - Arduino Digital 2
* Bluetooth RX-I - Arduino Digital 3
* Bluetooth RTS-O - nu se conectează

1. **Modulul Aplicație Pacient-SMARTPHONE (APS)**
   1. **Bluetooth**

Platforma Android include suport pentru Bluetooth care permite unui dispozitiv să facă schimb de date wireless cu alte dispozitive Bluetooth.

API-urile Android Bluetooth permit aplicațiilor să se conecteze fără fir la alte dispozitive Bluetooth.

Pachetele importate pentru Android si Bluetooth:

* import android.app.Activity;
* import android.bluetooth.BluetoothAdapter;
* import android.content.Intent;
* import android.os.Bundle;
* import android.view.View;

Din pachetul Bluetooth API-urile de baza sunt:

* BluetoothAdapter - reprezintă adaptorul local Bluetooth

**private BluetoothAdapter \_bluetooth = BluetoothAdapter.getDefaultAdapter();**

**Intent enabler = new Intent (BluetoothAdapter.ACTION\_REQUEST\_ENABLE); \_bluetooth.enable();**

* BluetoothDevice - reprezintă un dispozitiv Bluetooth la distantă, care conține informații despre cum ar fi numele, adresa, clasa si starea de legare

**private BroadcastReceiver \_discoveryReceiver = new BroadcastReceiver()**

**private BroadcastReceiver \_foundReceiver = new BroadcastReceiver()**

* BluetoothSocket - reprezintă interfața pentru un socket Bluetooth (similară cu un Socket TCP)
* BluetoothServerSocket - reprezintă un socket de server deschis care ascultă pentru cererile primite (similar cu un TCP ServerSocket).

Server

**\_serverSocket=\_bluetooth.listenUsingRfcommWithServiceRecord(PROTOCOL\_SCHEME\_RFCOMM, UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666"));**

**private void shutdownServer() //STOP SERVER**

Client

**socket = device.createRfcommSocketToServiceRecord (UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666"));**

* BluetoothClass - descrie caracteristicile generale și capabilitățile unui dispozitiv Bluetooth.

**Android/BluetoothHelper.class**

Pentru a avea acces la Bluetooth în aplicație folosim 2 permisiuni:

**<uses-permission android:name="android.permission.BLUETOOTH" />**

**<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />**

* 1. **HTTP Requests**

Pentru a trimite request-uri trebuie adaugată prima data o permisiune pentru internet:

**android.permission.INTERNET**

Pachetele necesare activității pentru a trimite "GET request”

* import com.android.volley.Request;
* import com.android.volley.RequestQueue;
* import com.android.volley.Response;
* import com.android.volley.VolleyError;
* import com.android.volley.toolbox.StringRequest;
* import com.android.volley.toolbox.Volley;

Obținem o conexiune HttpURLConnection apelând URL #openConnection() și transferă rezultatul către HttpURLConnection.

**HttpURLConnection urlConnection=(HttpURLConnection) url.openConnection();**

Instanțele trebuie să fie configurate cu setDoOutput (true) dacă includ un Request. Datele se transmit prin scrierea în fluxul returnat de

**URLConnection.getOutputStream ().urlConnection.setDoOutput(true);**

**OutputStream out=new BufferedOutputStream(urlConnection.getOutputStream();**

Răspunsul poate fi citit din stream-ul returnat de URLConnection.getInputStream ().

**InputStream in = new BufferedInputStream(urlConnection.getInputStream());**

Odata ce răspunsul a fost citit, HttpURLConnection ar trebui închis prin apelul **deconectare ().** Deconectarea eliberează resursele deținute de o conexiune, astfel încât acestea să poată fi închise sau reutilizate.

Pentru a transmite un Request se folosesc urmatoarele funcții:

* Inițierea unui request:

**RequestQueue ExampleRequestQueue = Volley.newRequestQueue(this);**

* Crearea unui StringRequest:

**StringRequest stringRequest = new StringRequest(Request.Method.GET, url,new Response.Listener<String>()**

* Pentru crearea unui nou request se folosește funcția **add ()**
* Se executa functia **onResponse()** daca serverul răspunde
* Crearea unui Error Listener pentru posibilele erori **new Response.ErrorListener()**
* Funcția se execută în cazul apariției unei erori **onErrorResponse(VolleyError error)**
* Pentru anularea unui Request **cancel (),** iar pentru anularea tuturor request-urilor se folosește **onStop ()**

1. **Modulul Aplicatie Web**

HTTP folosește un model **message-based** prin care clientul (browserul) trimite un mesaj sub formă de cerere (**HTTP request**) către server, iar server-ul returnează un mesaj de răspuns (**HTTP response**). Protocolul este "connectionless".

Toate mesajele HTTP sunt alcătuite din câteva **headere** (fiecare pe o linie separată, urmată de un rand liber) și, opțional, de un mesaj. Fiecare header are urmatoarea structură:

* **Header-name: header value**

Prima linie a fiecarei cereri HTTP este alcatuită din trei elemente separate prin spații:

* **Metoda HTTP (sau verbul HTTP)**: Cea mai utilizată metoda este **GET** care preia o resursă de pe serverul web. Cererile **GET** nu au un mesaj, de aceea dupa headere nu există conținut.
* **URL-ul cerut**: URL-ul funcționează ca un nume pentru resursa care a fost solicitată împreună cu un șir de interogări (sau parametrii) pe care clientul le-a introdus. Șirul de interogare este reprezentat de semnul ? (exemplu: index.php?**id=11**).
* **Versiunea HTTP folosită**: Singurele versiuni de HTTP folosite de obicei sunt versiunea 1.0, versiunea 1.1 si, mai nou, versiunea 2.0, însa cele mai multe browsere folosesc 1.1 ca versiune implicită. Exista cateva diferențe intre specificațiile celor trei menționate mai sus, dar singura deosebire cand vine vorba de securitate web este ca o cerere HTTP/1.1 nu poate fi trimisă fară headerul **Host**.

In afară de cele trei menționate mai sus, putem observa cateva headere interesante în acestă cerere HTTP simplă:

* **Referer**: este folosit pentru a indica adresa URL de la care provine cererea.
* **User-Agent**: este folosit pentru a furniza informații despre browserul de pe care s-a generat cererea.
* **Host**: specifică hostname-ul care apare in URL-ul accesat.
* **Accept**: browserul trimite headerul "Accept" pentru a specifica ce tip de document este așteptat in raspuns.
* **Accept-encoding**: funcționeaza la fel ca header-ul "Accept", doar ca acesta se refera la tipul de encoding al conținutului, si nu la conținut in sine. In acest caz, browserul accepta doua tipuri de compresie: [gzip](https://en.wikipedia.org/wiki/gzip" \t "_blank) și [deflate](https://en.wikipedia.org/wiki/DEFLATE). Comprimarea conținutului de pe pagina va micșora timpul de încărcare al acesteia.

**Structuri de baze de date si fișiere**

Modulul Aplicatie Cloud deține o evidentă a datelor sistemului în interiorul unei baze de date Azure SQL DB.

Am stabilit ca structura acesteia sa aiba urmatoarele tabele:

Pacient:

* Id(number) PK
* Email(string)
* UserName(string)
* Pwd(string)
* CNP(number)
* Nume(string)
* Prenume(string)
* Adresa(string)
* Data Nasterii(Date)
* Nr Telefon(string)
* Varsta(number)
* Inaltime(number)
* Greutate(number)
* Medici(many to many relationship) FK
* WarningsList(one to many)
* RecommandationList(one to many)

Medic:

* Id(int) PK
* Email(string)
* UserName(string)
* Pwd(string)
* CNP(string)
* Nume(string)
* Prenume(string)
* GradeProfesionale(string)
* OrarCabinet(string)
* Nr Telefon(string)
* Specializare(string)
* Pacienti(one to many)

PacientSensorData:

* Id(int) PK
* PacientId(int) FK one to many
* TimeStamp(datetime)
* Temperature(float)
* Humidity(float)
* Pulse(float)
* EKG(float)

PacientWarnings:

* Id(int) PK
* PacientId(int) FK one to many
* MedicId(int) FK one to one
* TimeStamp(datetime)
* WarningMessage(string)

PacientRecommandations:

* Id(int) PK
* PacientId(int) FK one to many
* MedicId(int) FK one to one
* TimeStamp(datetime)
* RecomandationMessage(string)