HE-ARC

Projet PA

Physio Data Visualization

Auteurs :
Gander Laurent

Responsables :
Rizzotti Aïcha

29 janvier 2018

Application mobile Android sur la visualisation du stress
Il s'agit de trouver des solutions de visualisation sur mobile pour traduire à l'utilisateur des émotions comme le taux de stress par le biais d'un bracelet avec des capteurs physiologique telque l'EDA l'electrodermal activity, la variabilité cardiaque... etc Le travail consiste à faire la conception et l'implémentation de la chaine, des capteurs, le serveur et l'application sur mobile.

Table des matières

Table des figures

1 Introduction

Ce projet a été réalisé dans le cadre des cours de projet P3 et d'android, du cursus informatique de troisième année. L'objectif était de concevoir un programme fonctionnel et réalisable en un semestre. Le but principal est de présenter la méthode de réalisation, ainsi que le logiciel terminé. Le langage utilisé pour ce projet est le Java sur Android. Le but de mon application est de trouver des solutions de visualisation sur mobile pour traduire à l'utilisateur des émotions, comme le taux de stress par le biais d'un bracelet avec des capteurs physiologique tel que l'EDA l'electrodermal activity, la variabilité cardiaque. . . etc
Le travail consiste à faire la conception et l'implémentation de la chaine, des capteurs, le serveur et l'application sur mobile. La réalisation a commencé par une phase d'analyse. Certaines informations qu'il a fallu compléter par des recherches personnelles ont été transmises. Vient ensuite une phase de préconception durant laquelle il a fallu établir un graphique UML définissant l'implémentation, ainsi qu'un use case et un diagramme de séquence. Dans la phase suivante, le projet a été codé.
Le rapport se construit de la manière suivante. L'explication des différents termes et fondements du projet. Il y est ensuite expliqué le résultat obtenu, les difficultés encourues et finalement une comparaison avec les spécifications détaillées de départ.

2 Déroulement

Durant la phase de conception, un cahier des charges, des spécifications détaillées, un diagramme UML, un diagramme de cas d'utilisation et un diagramme de séquence ont été réalisés. Une phase de réalisation a suivi cette phase de conception. Durant celle-ci, l'application a été implémentée de façon à respecter au mieux les objectifs décidés durant la première phase, mais certains ajustements ont été nécessaires. La phase finale est la phase de test durant laquelle le bon fonctionnement du programme a été examiné et les éventuels problèmes corrigés.

2.1 Phase de conception

Pour la phase de conception, les points importants des spécifications détaillées ont été réfléchis. J'ai commencé par faire une recherche et comparer des applications qui permettaient de faire ce qui m'était demandé, afin de trouver ce qui serait bien de faire et ce qu'il ne faudrait pas faire.
Après avoir fais ce travail de recherche et en ayant discuté avec Madame Rizzotti, nous avons décidé de réaliser une application qui : - permet de recevoir des données provenant d'un bracelet de type Empatica. - permet d'introduire des données utilisateurs. - permet de calculer les battements de coeur avec la caméra. - permet d'enregister les données utilisateurs dans une base de données. - permet d'enregistrer les logs dans une base de données. - sois userfriendly.
Pour parfaire ces objectifs, et comme décrit dans le diagramme UML, nous avons choisi de créer la hiérarchie suivante :

- Activity
  - MeasuresDetailActivity
- Fragments
  - AnalyseFragment
  - HomeFragment
  - MeasuresFragment
  - PhysioDataVisualisationFragment
  - VisualisationFragment
- HeartBeat
  - HeartBeatView
  - ImageProcessing
  - PhysioDataActivity

- Logs
  - CallReceiver
  - DatabaseLog
  - Log
  - LogRepository
  - LogSchema
  - PhoneCallReceiver
  - PhoneSMSReceiver
  - SMSReceiver
- Measures
  - DatabaseMeasures
  - MeasuresRepository
  - Measures
  - MeasuresAdapter
  - MeasureSchema
  - MeasuresConnection
  - MeasureService
- MainActivity

2.2 Phase de réalisation

Durant la phase de réalisation, nos objectifs ont été suivis et atteints. Cependant, l'objectif qui consistait à recevoir les données du bracelet, n'a pas été réalisé. M. Noguera n'avait pas de données à envoyer. Concernant le déroulement de cette phase, j'ai commencé par m'occuper du design, puis j'ai codé les différentes classes et intégré les fonctionnalités correspondantes.

2.2.1 MainActivity

Mon activity principale contient un fragment qui sera changé suivant les actions de l'utilisateur. Le fragment qui est chargé de base est le HomeFragment. Un utilisateur peut changer de fragment en ouvrant le menu qui se trouve en haut à gauche, ce menu glisse pour prendre part en partie sur l'écran. L'utilisateur à la choix entre plusieurs options : - Home - Analyse - Measure - Visualisation
Chaque fragment est expliqué plus loin dans le rapport.
Pour pouvoir changer de fragment, j'ai overrider une fonction de la classe suivante

```
NavigationView.OnNavigationItemSelectedListener
```

La fonction est la suivante

```
public boolean onNavigationItemSelected(MenuItem item) {
    int id = item.getItemId();


    if (id == R.id.nav_home){
        HomeFragment homeFragment = new HomeFragment();
        getSupportFragmentManager().
        beginTransaction().
        replace(R.id.main_fragment ,homeFragment).commit();
        homeFragment.onMeasureServiceConnected();
    }
    if (id == R.id.nav_visualisation) {
        VisualisationFragment visualisationFragment =
        new VisualisationFragment();

        getSupportFragmentManager().
        beginTransaction().
        replace(R.id.main_fragment ,visualisationFragment).commit();
```

```
            visualisationFragment.onMeasureServiceConnected();

        } else if (id == R.id.nav_analyse) {
            AnalyseFragment analyseFragment = new AnalyseFragment();
            getSupportFragmentManager().
            beginTransaction().
            replace(R.id.main_fragment, analyseFragment).commit();
            analyseFragment.onMeasureServiceConnected();

        } else if (id == R.id.nav_mesures) {
            MeasuresFragment mesuresFragment = new MeasuresFragment();
            getSupportFragmentManager().
            beginTransaction().
            replace(R.id.main_fragment ,mesuresFragment).commit();
            mesuresFragment.onMeasureServiceConnected();
        } else if (id == R.id.nav_share) {

        }

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        drawer.closeDrawer(GravityCompat.START);
        return true;
    }
```

Mon activity commence déjà par vérifier si j'ai l'autorisation d'utiliser les capteurs qui seront utiles à mon application.

- Camera
- Appel entrant, sortant et manqué (READ_PHONE_STATE)
- SMS reçu (SMS_RECEIVE)

Ces vérifications se font de la manière suivante

```
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) ==
        PackageManager.PERMISSION_DENIED)
            {
                ActivityCompat.requestPermissions( this, new String[]
                {Manifest.permission.CAMERA}, 1);
            }

    if (ContextCompat.checkSelfPermission(this,Manifest.permission.READ_PHONE_STATE) ==
        PackageManager.PERMISSION_DENIED)
            {
                ActivityCompat.requestPermissions( this, new String[]
                {Manifest.permission.READ_PHONE_STATE}, 1);
            }

    if (ContextCompat.checkSelfPermission(this, Manifest.permission.RECEIVE_SMS) ==
        PackageManager.PERMISSION_DENIED)
          {
              ActivityCompat.requestPermissions( this, new String[]
              {Manifest.permission.RECEIVE_SMS}, 1);
          }
```

Si l'application n'a pas l'autorisation, alors elle demandera à l'utilisateur de la lui donner.
Il faut aussi ajouter ces autorisations dans le manifest.xml
Que contient un manifest ?
Le fichier manifest permet de décrire votre application. On y retrouve :

- le nom du package de l'application. Il servira d'identifiant unique.
- tous les composants de l'application (Activity, Services, BroadCast Receivers, Content providers).
- on détermine dans quels processus les composants de l'application seront contenus.
- les permissions nécessaires pour le bon fonctionnement de l'application
- les informations contenant les versions de l'Android API requis pour exécuter votre application

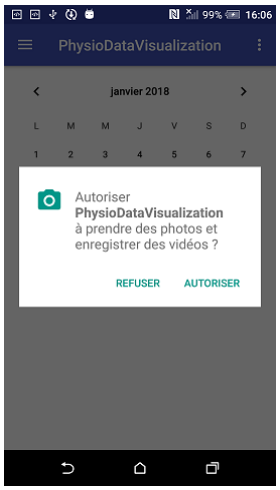

FIG. 1 : Demande de permission pour la caméra

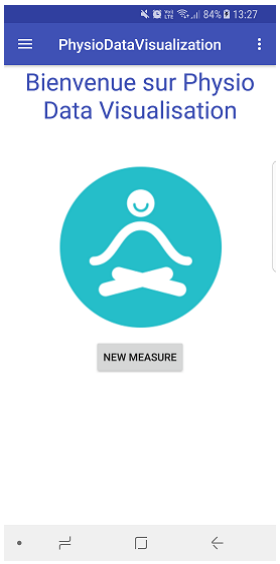

FIG. 2 : Home fragment

- les bibliothèques utilisées par votre application.

Ensuite, je crée un service qui me permet d'interagir avec ma base de données DatabaseMeasures MeasureService recupère une Instance de ma base de données.Je crée un objet MeasuresRepository. Je passe une Instance de ma base à MeasuresRepository. Comme je l'expliquerai plus loin dans le rapport, MeasuresRepository me permet d'interagir avec ma base de données.

2.2.2 HomeFragment

Ce fragment apparaît à l'ouverture de l'application. Il contient une page d'accueil et un bouton qui permet de faire une nouvelle mesure.

2.2.3 MeasuresFragment

Ce fragment permet l'ajout de données utilisateurs et de faire une nouvelle mesure. On peut y entrer plusieurs paramètres : - L'index cénesthésique, c'est à dire la façon dont on se sent. - les heures de sommeil. - la position dans laquelle on fait la mesure. - un commentaire.
Dès que toutes les données sont rentrées, on peut appuyer sur le bouton 'start measurement', en cliquant dessus, on ouvre PhysioDataActivity.

```
final Intent intent = new Intent( getActivity(), PhysioDataActivity.class);
```

```
startActivityForResult( intent , 1 );
```

on recupère le resultat avec la fonction

```
public void onActivityResult(int requestCode, int resulCode, Intent data)
{
    super.onActivityResult( requestCode, resulCode,data );
    Date date = new Date();
    if(1 == requestCode)
    {
        avgHeartBeat = data.getIntExtra( "Heart_Beat", 0 );
        int stressIndex = calculStressIndex(index, sommeil, avgHeartBeat);
        ((MainActivity)getActivity()).getService().createMeasure
        (index, sommeil, position, comment, avgHeartBeat, stressIndex, date);
    }
}
```

2.2.4 PhysioDataActivity

Cette activité nous permet de mesurer notre rythme cardiaque en utilisant la caméra. L'utilisateur va poser son doigt sur la caméra, celle-ci va capturer en boucle des images de notre doigt et ainsi comparer le taux de rouge dans l'image. A chaque battement de coeur, il y aura un afflux de sang de sang dans notre doigt, il deviendra ainsi plus rouge. En incrementant une variable à chaque changement de couleur, on trouve notre rythme cardiaque.
on renvoie le résultat à notre fragment avec

```
public void onBackPressed()
{
    int avg = 0;

    for(int i = 0; i< listMeasures.size(); i++)
    {
        avg += listMeasures.get( i );
    }

    Intent intent = new Intent( );

    intent.putExtra( "Heart_Beat", avg/listMeasures.size() );
    setResult(0, intent);
    finish();
}
```

2.2.5 HeartBeatView

Permet d'afficher la view qu'on a l'écran, qui contient : - canvas - image rouge s'il y a un battement de coeur - image verte s'il n'y a pas de de battement de coeur

2.2.6 ImageProcessing

C'est dans cette classe que tout le calcul sur la couleur des pixels se fait.

2.2.7 AnalyseFragment

Ce fragment contient : - une SearchView qui permet à l'utilisateur de rechercher une mesures par rapport à une date. On entre une date dans la barre de recherche et on va chercher le filter dans la classe MeasuresAdapter qui nous retourne les valeurs correspondantes à la date. - une ListView de mesures triées par date.
Quand on clique sur un élément de la ListView, on ouvre une nouvelle Activity (MeasuresDetailActivity) grâce à un AdapterView. On envoie la mesure via un Intent. Pour cela, j'ai dû sérialiser ma classe Mesures. La ListView a un adapter qui lui est setter. Un Adapteur est un objet qui gère les données, c'est comme un intermédiaire entre les données et et la vue qui les représente.

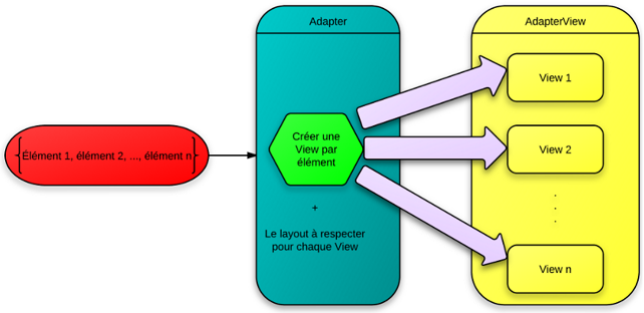

FIG. 3 : Adapter

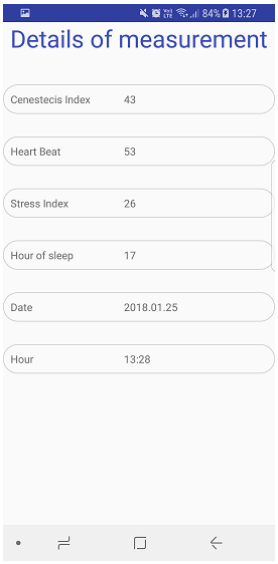

FIG. 4 : Details de la mesure

2.2.8 MeasuresDetailActivity

Cette activity permet d'afficher toutes les informations da la mesure sur laquelle on a cliqué dans le AnalyseFragment. Elle recoit un Intent qui contient la mesure et affiche ces informations. - Indice cénéstésique - Rythme cardiaque - Indice de stress - Les heures de sommeil - La date - L'heure

2.2.9 MeasuresAdapter

Comme expliqué précédemment, un adapter permet de gérer les données. La classe MeasuresAdapter permet de récolter les mesures effectués et de les retourner au AnalyseFragment pour qu'elles soient affichées. Au préalable, on aura instancier un objet Filter qui nous permet de filtrer les mesures par dates et de retourner la liste filtrée.
Dans ce fichier nous avons une autre classe, MeasuresHolder, qui nous permet de choisir ce qui sera affiché dans la liste. Personnellement, j'ai choisi d'afficher la position dans laquelle la mesure a été faites, la date, ainsi que la l'indice de stress.

2.2.10 VisualisationFragment

Ce fragment permet à l'utilisateur de visualiser son stress. Il lui est possible de choisir la date et il verra sur un graphe le taux de stress qu'il a eu à un moment de la journée.
Pour réaliser ce fragment, j'ai récupéré les mesures contenues dans ma base de données grâce à mon MeasureService. Je stocke toutes les dates dans un Spinner en utilisant un ArrayAdapter. L'ArrayAdapter prend en paramètre le context, un layout et une liste qui contient les dates. Le Spinner permet de choisir une date, par rapport à la date choisie, le graphe affiche la valeur de stress à l'heure de la mesure.

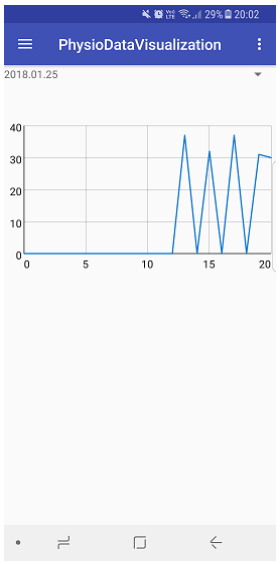

FIG. 5 : Graphe

2.2.11 DatabaseMeasures

Cette classe contient les informations liées à la base de données.
Elle permet : - de créer de la base de données. - de mettre à jour la base de données.
Elle contient le nom de la base de données, ainsi que la version. Pour la base de données, la classe hétite de SQLiteOpenHelper. SQLiteOpenHelper permet de gérer la base de données, ainsi que la version.

```
public class DataBaseMeasures extends SQLiteOpenHelper {

    public static  DataBaseMeasures sInstance;

    public static final String DATABASE_NAME = "measures.db";

    public DataBaseMeasures(Context context) {

        super( context, DATABASE_NAME, null, 9);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL( MeasureSchema.CREATE_TABLE_QUERY);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL( MeasureSchema.DROP_TABLE_QUERY );
        onCreate( db );
    }

    public static synchronized DataBaseMeasures getInstance(Context context)
    {
        if (sInstance == null)
        {
            sInstance = new DataBaseMeasures( context.getApplicationContext() );
        }
        return sInstance;
    }
}
```

La fonction getInstance me permet de m'assurer que je n'ai qu'une seule et unique instance de ma base de données. Ceci évite les problèmes de connexion multiple.

2.2.12 MeasureSchema

Cette interface me permet de construire la table dans ma base de données.

On commence par définir le nom des colonnes qu'on veut dans notre table.

```
    String TABLE_NAME = "measures";
    String COLUMNS_1 = "ID";
    String COLUMNS_2 = "CYNESTESIS_INDEX";
    String COLUMNS_3 = "HOURS_SLEEP";
    String COLUMNS_4 = "POSITION";
    String COLUMNS_5 = "COMMENTS";
    String COLUMNS_6 = "HEARTBEAT";
    String COLUMNS_7 = "STRESS_INDEX";
    String COLUMNS_8 = "DATE_MEASURES";
    String COLUMNS_9 = "HOURS_MEASURES";


    String[] tabColumns = new String[]{COLUMNS_1, COLUMNS_2, COLUMNS_3, COLUMNS_4,
        COLUMNS_5, COLUMNS_6, COLUMNS_7, COLUMNS_8, COLUMNS_9};
```

Puis, on définit une variable de type string qui contient la commande SQL qui permet de créer la table

```
    String CREATE_TABLE_QUERY = "CREATE TABLE IF NOT EXISTS "
            + TABLE_NAME
            + " (" + COLUMNS_1
            + " INTEGER PRIMARY KEY AUTOINCREMENT,"
            + COLUMNS_2
            + " INTEGER, "
            + COLUMNS_3
            + " INTEGER,"
            + COLUMNS_4
            + " INTEGER,"
            + COLUMNS_5
            + " TEXT,"
            + COLUMNS_6
            + " INTEGER, "
            + COLUMNS_7
            + " INTEGER, "
            + COLUMNS_8
            + " TEXT, "
            + COLUMNS_9
            + " TEXT)";
```

Et pour finir, on créer une variable de type string pour la suppression de la table.

```
    String DROP_TABLE_QUERY = "DROP TABLE IF EXISTS " + TABLE_NAME;
```

2.2.13 MeasureService

Qu'est ce qu'un service ? >Tout comme les activités, les services possèdent un cycle de vie ainsi qu'un contexte qui contient des informations spécifiques sur l'application et qui constitue une interface de communication avec le restant du système. Ainsi, on peut dire que les services sont des composants très proches des activités (et beaucoup moins des receivers, qui eux ne possèdent pas de contexte). Cette configuration leur prodigue la même grande flexibilité que les activités. En revanche, à l'opposé des activités, les services ne possèdent pas d'interface graphique : c'est pourquoi on les utilise pour effectuer des travaux d'arrière-plan. source : https ://openclassrooms.com/courses/creez-des-applications-pour-android/les-services-3
Ce service permet de créer le lien entre mon application et la base de données. Depuis mon MainActivity, je récupère un objet Service. Depuis ce moment, je peux intéragir avec mes données.
MeasureService instancie un objet de type MeasuresRepository.

```
public IBinder onBind(Intent intent) {

        DataBaseMeasures dataBaseMeasures = DataBaseMeasures.getInstance( this );

        database = dataBaseMeasures.getWritableDatabase();
        measureRepository = new MeasureRepository(database);
        return binder;
    }
```

Et maintenant je peux appeler les methodes liées à la base de données qui sont contenues dans MeasureRepository.

```
public List<Measures> findMeasures(){

        return measureRepository.getAllListMeasures();
    }

    public void createMeasure(int progress, int seekBarSommeilProgress, Integer position,
         String comment, int heartBeat, int stressIndex, Date date)
    {
        measureRepository.create(progress, seekBarSommeilProgress, position, comment,
            heartBeat ,stressIndex, date);
        Intent intent = new Intent(ACTION_CREATE_MEASURE);
        sendBroadcast(intent);
    }
```

2.2.14 MeasuresRepository

Cette classe contient les méthodes qui permettent d'intéragir avec la base de données. - create - select

```
public List<Measures> getAllListMeasures()
    {
        List<Measures> measureList = new ArrayList<Measures>();
        Cursor res = db.query(TABLE_NAME, tabColumns, null, null, null, null, COLUMNS_1);
        if (res != null)
        {
            res.moveToFirst();
            while(!res.isAfterLast())
            {
                String date = res.getString(7) + " " + res.getString( 8 );
                measureList.add(new Measures(res.getInt(1),res.getInt(2),res.getInt(3),
                    res.getString(4), res.getInt(5), res.getInt(6), date));
                res.moveToNext();
            }
        }
        res.close();
        return measureList;
    }

    public void create(int progress, int seekBarSommeilProgress, Integer position, String
         comment, int heartBeat, int stressIndex, Date date){
        ContentValues contentValues = new ContentValues();
        contentValues.put(COLUMNS_2, progress);
        contentValues.put(COLUMNS_3,seekBarSommeilProgress);
        contentValues.put(COLUMNS_4,position);
        contentValues.put(COLUMNS_5,comment);
        contentValues.put(COLUMNS_6, heartBeat);
        contentValues.put(COLUMNS_7, stressIndex);
        contentValues.put(COLUMNS_8, DateFormat.format("yyyy.MM.dd",date).toString());
        contentValues.put(COLUMNS_9, DateFormat.format( "HH:mm:ss", date).toString());

        long measureId = db.insert( TABLE_NAME, null, contentValues );

    }
```

Ce principe est reproduit pour la base de données des Logs.
Je vais expliquer comment on récupère les événements : - Appel entrant - Appel sortant - Appel manqué - SMS entrant

2.2.15 Log

Les deux classe, PhoneCallReceiver et SMSReceiver, héritent de BroadcastReceiver. En héritant de BroadcastReceiver et en implémentant la fonction onReceive(), on peut facilement recevoir l'événement et agir en fonction de celui-ci.

Appels On récupère l'événement qu'on stocke dans une variable state. Ensuite, suivant la valeur de state, on appelle les fonctions qui ajouteront l'événement correspondant au state dans la base de données.

```
public void onReceive(Context context, Intent intent) {
        if(intent.getAction().equals("android.intent.action.NEW_OUTGOING_CALL")){
        }
```

```
        else{
            String stateStr = intent.getExtras().getString(TelephonyManager.EXTRA_STATE);
            int state = 0;
            if(stateStr.equals(TelephonyManager.EXTRA_STATE_IDLE)){
                state = TelephonyManager.CALL_STATE_IDLE;
            }
            else if(stateStr.equals(TelephonyManager.EXTRA_STATE_OFFHOOK)){
                state = TelephonyManager.CALL_STATE_OFFHOOK;
            }
            else if(stateStr.equals(TelephonyManager.EXTRA_STATE_RINGING)){
                state = TelephonyManager.CALL_STATE_RINGING;
            }

            onCallStateChanged(context, state);
        }
    }

    protected void onIncomingCallStarted(Context ctx, Date start){}
    protected void onOutgoingCallStarted(Context context, Date start){}
    protected void onMissedCall(Context ctx, Date start){}

    public void onCallStateChanged(Context context, int state) {
        if (lastState == state) {
            //No change, debounce extras
            return;
        }
        switch (state) {
            case TelephonyManager.CALL_STATE_RINGING:
                callStartTime = new Date();
                onIncomingCallStarted(context, callStartTime);
                break;
            case TelephonyManager.CALL_STATE_OFFHOOK:
                //Transition of ringing->offhook are pickups of incoming calls.  Nothing
                    done on them
                if (lastState != TelephonyManager.CALL_STATE_RINGING) {
                    callStartTime = new Date();
                    onOutgoingCallStarted(context, callStartTime);
                }
                break;
            case TelephonyManager.CALL_STATE_IDLE:
                //Went to idle-  this is the end of a call.  What type depends on
                    previous state(s)
                if (lastState == TelephonyManager.CALL_STATE_RINGING) {
                    //Ring but no pickup-  a miss
                    onMissedCall(context, callStartTime);
                }
                break;
        }
        lastState = state;
    }
```

SMS Le méchanisme est le même pour le SMS, il est même plus simple vu qu'on a qu'un événement qui est la réception du message.

```
@Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(ACTION_RECEIVE_SMS)) {
            Bundle bundle = intent.getExtras();
            if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");

                final SmsMessage[] messages = new SmsMessage[pdus.length];
                for (int i = 0; i < pdus.length; i++) {
                    messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                }
                if (messages.length > -1) {
                    Date date = new Date();
                    onIncomingSMS(context, date);
                }
            }
        }
    }
```

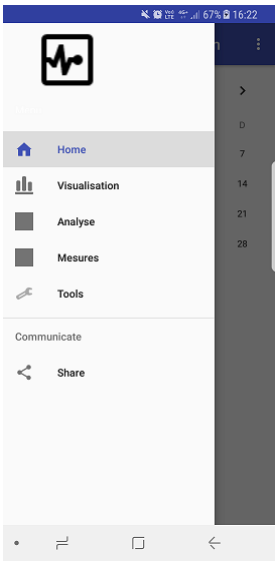

FIG. 6 : Menu gauche

3 Résultats obtenus

A l'ouverture de l'application, on a un fragment qui s'ouvre avec un calendrier. On peut naviguer entre les fragments en utilisant le menu qui se trouve en haut à gauche.
On peut ajouter une nouvelle mesure en cliquant sur 'Measures', l'ajout d'une mesure se passe de la manière suivante

- Entrer l'index cénéstésique
- Entrer les heures de sommeil de la nuit précédente
- Entrer la position dans laquelle on est pendant la mesures
  - Assis
  - Couché
  - Debout
- Entrer un commentaire
- Cliquer sur le bouton 'start measurement'

Une nouvelle activity s'ouvre et on pose notre doigt sur la camera et on attend sans bouger que l'application calcule notre rythme cardiaque.
Quand ceci est fait, on peut cliquer sur le bouton retour de notre application et la mesure est enregistrée dans la base de donnnées.
On peut aller visualiser les mesures prises en allant sur 'Visualisation' dans notre menu. On peut choisir la date dans une liste déroulante qui se trouve en haut à gauche. Un graphe représente le niveau de stress à l'heure où la mesure a été faite.
Pour voir toutes les mesures, on peut aller sur 'Analyse' qui se trouve dans le menu gauche. Le fragment contient la liste de toutes les mesures et affiche : - La position - La date - L'indice de stress
En cliquant sur une mesure, on ouvrira une nouvelle activity qui affichera la mesure plus en détail. Les données sont passées au travers d'un Intent, en utilisant la fonction startActivity(intent) Les informations affichées sont : - Indice cénéstésique - Rythme cardiaque - Indice de stress - Les heures de sommeil - La date - L'heure

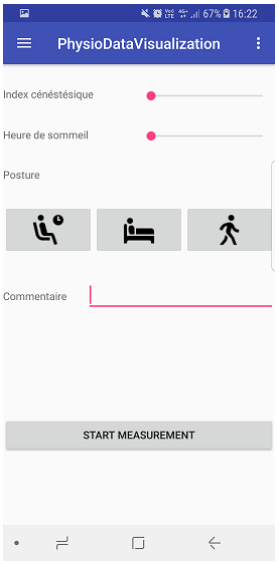

Fig. 7 : Mesures

Fig. 8 : Heart

Fig. 9 : Analyse

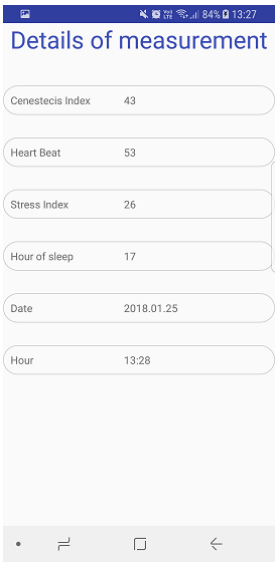

Fig. 10 : Details des mesures

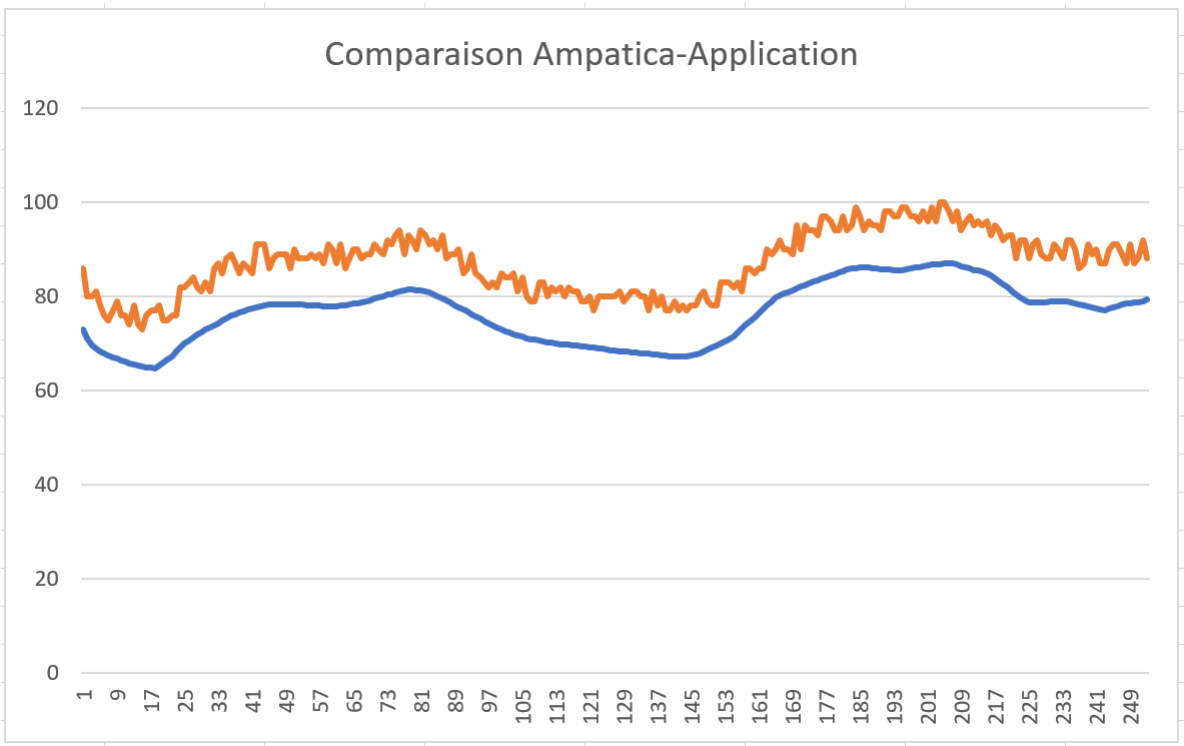

FIG. 11 : Graphe qui compare les résultats de l'application et ceux de l'Ampatica

4 Comparaison des résultats

Avec l'aide de Dominique Ducommun, j'ai effectué en parallèle des mesures sur le bracelet Ampatica et sur l'application. Dominique m'a envoyé les informations récoltées sur l'Ampatica, dont le heart beat, et j'ai pu comparer les résultats en faisant un graphe des mesures.
En orange, nous avons les mesures de l'application et en bleu, les mesures de l'Ampatica.
Commme nous pouvons le constater, les deux mesures suivent la même courbe, mais avec un offset entre 5 et 15 environ. On peut en déduire que l'application retourne des résultats qui sont bons. Après on peut voir que la courbe de mesures de l'application est moins lisse que la courbe de l'Ampatica. Ceci s'explique par le fait qu'on pose le doigt sur la caméra, le mesure peut être tronqué par le mouvement du doigt de l'utilisateur sur la caméra, ce qui va influencer le taux de la couleur rouge récupéré.

5 Tests

La phase de tests s'est déroulée depuis la création de la première version fonctionnelle jusqu'à la rédaction du rapport, soit un suivi constant de l'état de fonctionnement du programme. Le code n'a pas été testé via des tests unitaires ou d'implémentation, mais un test de fonctionnalité a été réalisé.
Mon application a été utilisée par de tierce personnes qui m'ont donné certains retours - Design pas excellent - Calendrier inutile - Facile d'utilisation

6 Difficultés rencontrées

La plus grande difficulté au début du projet a été de comprendre le fonctionnement d'Androïd et de structurer l'application de la bonne manière. Ensuite, j'ai eu un problème avec la base de données des mesures. Quand je changeais la structure de ma table, mes insertions ne marchaient plus, car la table ne se mettait pas à jour. Le suppression de l'application sur le téléphone ne supprime pas la base de données donc le problème était toujours le même. Pour pouvoir pallier à ce problème, il faut incrementer la version de la base de données dans DatabaseMeasures.

7 Améliorations possibles

Certaines améliorations peuvent être apportées à cette application. Effectivement, étant un projet de semestre, l'application n'a pas pu être fignolée. La première amélioration est le design, comme vous pouvez le constater, l'application peut être beaucoup améliorer de ce point là. La deuxième amélioration serait de pouvoir recevoir les donnnées de M. Noguera et de pouvoir les comparer avec les données de l'application. Un troisième point serait que l'utilisateur recoive une notification quand il est stressé.
Pour l'application, j'ai créée des base de données, une pour les mesures et une pour les logs, on pourrait ne faire que une base de données avec deux tables différentes.

8 Conclusion

Les différentes étapes de conception ont été réalisées correctement. La problématique a rapidement été cernée et les spécifications détaillées réalisées. Il a été possible de définir ces dernières après avoir déterminé les points chauds. La phase de réalisation a suivi les spécifications détaillées mais des changements ont dû être appliqués au vu du déroulement du projet.
Les objectifs principaux ont quant à eux été atteints avec succès. Comme on peut le voir au chapitre précédant, l'application est fonctionnelle. Cependant, une phase de test finale et générale n'a pas pu être mise en place par manque de temps.
Suite à la phase de réalisation et aux tests, il a été confirmé que l'application était fonctionnelle et qu'elle correspondait au cahier des charges et aux spécifications détaillées. L'une des perspectives d'amélioration du projet serait la réalisation d'une connexion avec un serveur pour récupérer les données générées par M. Noguera.

9 Annexes

9.1 Cahier des charges

Voici mon cahier des charges qui a été définit avec Madame Rizzotti ### Cadre du projet Ce projet est réalisé dans le cadre du cours Projet PA de 3ème années de bachelor en informatique, orientation DLM, au sein de la HE-Arc. Le projet est à réaliser individuellement et le sujet a été choisi parmis une liste de projet. Le logiciel est programmé en Java et est développé sur Android Studio.

9.1.1 Principe de l'application

Le projet que je dois réaliser est une application Android, ce projet est en liens avec le projet de Guillaume Noguera. Le but de ce projet est de trouver des solutions de visualisation sur mobile pour traduire à l'utilisateur des émotions comme le taux de stress par le biais d'un bracelet avec des capteurs physiologique tel que l'EDA l'Electrodermal Activity, la variabilité cardiaque, etc...
L'utilisateur pourra entrer l'état dans lequel il se sent, c'est-à-dire : - Son cénesthésique (état de bien être personnel). - Ses heures de sommeil. - Un commentaire. - Le moment de la journée. - Une source de stress. Guillaume Noguera s'occupe d'analyser des données physiologiques recueillis depuis le bracelet, puis transmet à l'application un taux de stress échelonné entre 1 et 5. L'utilisateur pourra visualiser ce taux de stress depuis l'application. L'utilisateur pourra aussi prendre son ń taux de stress ż via les capteurs de son téléphone en posant son doigt sur le capteur qui se trouve à l'arrière de l'appareil. L'utilisateur pourra comparer ses sentiments de bien être avec les résultats du bracelet et avec ceux du capteur.

9.1.2 Objectifs

Objectifs principaux :

- Recevoir et visualiser l'état de stress envoyé par le bracelet
- Pouvoir entrer des informations en relation avec l'acquisition des données
- Avoir une utilisation user friendly
- Sauvegarde des données

Objectifs secondaires :

- Utiliser les capteurs à l'arrière du téléphone pour le prise du stress
- Comparer les résultats
- Alerte si la personne est stressée
- Log (données entrantes)