



RÉSEAUX SANS FIL

U.E SYSTÈME DE TÉLÉCOMMUNICATIONS SANS FIL ET MOBILE - 2

Rapport sur le projet réseaux sans fil

AUTEURS :

MATHIS SIGIER LUCAS THIETART THOMAS GRUGET HUGO LE CLAINCHE

DÉPARTEMENT SCIENCES DU NUMÉRIQUE
DEUXIÈME ANNÉE
ENSEEIHT

ANNÉE 2023-2024

Table des matières

1	Introduction	1
2	Côté mobile	1
2.1	L'accéléromètre	1
2.2	Construction du JSON envoyé au serveur	2
2.3	La connexion avec le serveur	3
2.4	Mise en place de l'application	4
3	Côté serveur	4
4	Code de Lambda	5
5	Traitement des Mesures avec une FFT	8
5.1	Validation des Données	8
5.2	Prétraitement des Données	8
5.3	Calcul de la FFT	8
5.4	Calcul des Magnitudes	8
5.5	Identification de la Fréquence Fondamentale	8
5.6	Calcul de la Cadence et des Pas	8
5.7	Retour des Résultats	9
6	Conclusion	9

1 Introduction

Ce projet a pour but de nous familiariser avec la gestion de données de capteurs, leur transmission via un réseau sans fil et leur traitement sur une plateforme cloud. Nous créerons un podomètre, une application qui compte les pas de l'utilisateur. Le smartphone servira de dispositif de collecte de données, connecté à un point d'accès wifi, tandis que le traitement des données, incluant le calcul des pas, sera effectué par une application web sur un serveur.

Concrètement, l'utilisateur possède un smartphone avec une application mobile qui recueille régulièrement les données de capteurs et les transmet à une plateforme cloud pour traitement. Celle-ci calcule en temps réel le nombre de pas de chaque utilisateur. Dans ce projet, nous nous concentrerons uniquement sur le comptage de pas, sans le suivi de la position. Notre mission est de développer l'application mobile et la plateforme de traitement décentralisée, en utilisant une connexion wifi simplifiée. L'application mobile sera conçue pour Android et la partie distante sera une application web (aussi appelée serveur par la suite).

2 Côté mobile

L'appareil mobile joue le rôle d'accéléromètre et rapporte au serveur les données récoltées pour calculer le nombre de pas sur chaque ensemble d'échantillons.

Après le calcul et l'envoi du nombre de pas par le serveur, le mobile le reçoit et incrémente le nombre de pas total.

2.1 L'accéléromètre

Le téléphone est doté d'un accéléromètre qui fonctionne à la cadence $T_e = \frac{1}{f_e} = 20ms$ (soit $f_e = 50Hz$). Cette période d'échantillonnage est déjà enregistrée dans le téléphone Android pour le management du capteur d'accélération. C'est la deuxième cadence la plus rapide, la première étant une cadence indiquée de $0ms$, ce qui n'était pas pratique pour les calculs de taille d'échantillons (calcul de la fréquence impossible).

On envoie au serveur un objet JSON qui contient $N_{sample} = 750$ échantillons d'accélération. Ainsi, on envoie toutes les $t_{chantillonnage} = \frac{N_{sample}}{f_e} = 15$ secondes N_{sample} échantillons d'accélération.

Les choix de ces paramètres a du être revu car initialement le calcul de la fft au sein du serveur n'était pas assez précis : la fréquence d'échantillonnage et la taille des échantillons n'étaient pas assez grands.

On précisera ce sujet dans la partie ?? qui traite du calcul du nombre de pas par le serveur.

2.2 Construction du JSON envoyé au serveur

Dans cette partie, nous allons expliquer comment est construit le JSON de N_{sample} échantillons d'accélération.

Dès que l'accéléromètre enregistre une valeur d'accélération, la méthode `onSensorChanged`, écrite ci-dessous est appelée.

```
@SuppressWarnings("SetTextI18n")
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        float z = event.values[2]; // Accélération verticale
        verticalAcceleration.add(z);
        int taille_samples = verticalAcceleration.size();
        String s_acceleration_instant = String.valueOf(z);
        description.setText("Accélération instantanée (axe z) : "
            + s_acceleration_instant + " m/s2");
        if (isSendingContinue) {
            // On envoie la donnée au serveur
            sample++;
            if (sample == NBmaxSample) {
                // Conversion de l'accélération en JSON
                JSONObject json = new JSONObject();
                try {
                    JSONArray jsonArrayEnvoie = new JSONArray();
                    for (Float zi : verticalAcceleration) {
                        jsonArrayEnvoie.put(zi);
                    }
                    json = new JSONObject().put("name: \"accelerations\", jsonArrayEnvoie);

                    System.out.println(json.toString());
                    sendPostJSON(json);
                } catch (JSONException e) {
                    throw new RuntimeException(e);
                }
                sample = 0;
                verticalAcceleration.clear();
            }
        } else {
            description.setTextColor(Color.BLUE);
            sample = 0;
        }
    }
}
```

FIGURE 1 – Construction du JSONObject de N_{sample} échantillons qui va être envoyé via une requête HTTP au serveur

Dans le code de la MainActivity, `verticalAcceleration` est la liste de échantillons d'accélération. Dès que le capteur capture une accélération, elle est enregistrée dans la liste `verticalAcceleration`.

Si le mode d'envoi continue est activé (à travers le boolean `isSendingContinue`), on incrémente l'indice `sample`. Sinon on le met à 0.

Quand le nombre d'échantillon vaut N_{sample} , on peut procéder à la construction du JSONObject que l'on envoie au serveur. Pour cela, on construit un JSONArray avec toutes les valeurs d'accélération et on l'associe à la clé "accelerations" dans le JSONObject pour que le serveur reconnaisse qu'on lui envoie bien des données d'accélération.

Une fois que le JSONObject est créé, on l'envoie au serveur en appelant la méthode `sendPostJSON` qui est détaillée figure 6.

2.3 La connexion avec le serveur

L'appareil mobile envoie ses via la protocole HTTP. Dès que l'accéléromètre a capturé N_{sample} échantillons, la méthode `sendPostJSON` est appelée (code de la figure 6). Elle prend en paramètre un JSONObject et crée la requête HTTP qui va être envoyée au serveur.

```
private void sendPostJSON(JSONObject json) {
    Log.d( tag: "sendPostJSON", msg: "Début SPJSON Envoie de la donnée : " + json);
    JSONObjectRequest stringRequest = new JSONObjectRequest(Request.Method.POST, url.toString(), json,
        new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject s) {
                Log.d( tag: "sendPostJSON", msg: "Envoie effectué");
                Log.d( tag: "sendPostJSON", msg: "Réponse : " + s);
                //Toast.makeText(podoContext, s.toString(), Toast.LENGTH_SHORT).show();
                //bouton_envoie_serveur_continue.setBackgroundColor(Color.GREEN);

                String body = null;
                int pasMesure;
                try {
                    body = s.getString( name: "body");
                    JSONObject jsonBody = new JSONObject(body);
                    pasMesure = jsonBody.getInt( name: "steps");
                } catch (JSONException e) {
                    throw new RuntimeException(e);
                }

                nombrePas += pasMesure;
                majViewPas();
            }
        }, new Response.ErrorListener() {
            no usages
            @Override
            public void onErrorResponse(VolleyError volleyError) {
                Log.e( tag: "sendPostJSON", msg: "ERROR SEND POST" + volleyError.getMessage());
            }
        }
    );
}
```

FIGURE 2 – Méthode `sendPostJSON` appelé lorsque l'accéléromètre a capturé N_{sample} échantillons

Dans cette méthode, on crée une requête HTTP POST car on envoie dont le JSONObject est le contenu. On renseigne l'URL du serveur dans la requête.

Lorsque l'envoi échoue, un message d'erreur est affiché dans les logs. Sinon, le serveur envoie sa réponse à travers un `JSONObject`.

Une fois le `JSONObject` reçu, la méthode **onResponse** est appelée (elle est directement implantée dans la requête HTTP (figure 6)). Elle prend en paramètre le `JSONObject s` du serveur, dont on récupère le nombre de pas calculé par le serveur. On l'additionne au nombre total de pas qui est compté par la variable globale *nombrePas*.

Puis, on met à jour le nombre de pas affiché à l'écran à travers la méthode `majViewPas()`.

2.4 Mise en place de l'application

Lors de l'exécution de l'application, il y a plusieurs champs : un bouton pour commencer à envoyer les blocs d'échantillons au serveur (et donc calculer le nombre de pas), un champs textuel pour le nombre de pas cumulé, et un bouton pour réinitialiser le nombre de pas.

Le champ textuel du nombre de pas cumulé est mis à jour en fonction du nombre de pas envoyé et calculé par le serveur.


3 Côté serveur

L'objectif de l'application Web est d'analyser l'échantillons d'accélération envoyés par l'application mobile pour calculer le nombre de pas.

Pour cela on calcule la cadence, c'est à dire le nombre de pas par seconde en analysant la fréquence fondamentale du signal envoyée en appliquant la transformée de Fourier à celui-ci. En multipliant la cadence par la durée d'échantillonnage $t_{\text{chantillonnage}}$.

Nous utilisons alors AWS pour héberger notre API. On déclare dans API Gateway les différentes communications, ici nous prendrons en compte uniquement la méthode POST qui envoie les accélérations verticales. Lors que notre Gateway reçoit ce POST, on lance "appPodometer_{calcul}" la fonction qui traite ce post.

Déclencheur


API Gateway: [app_Podometre](#)
arn:aws:execute-api:eu-north-1:637423371000:wd852cn056/*/POST/step
Point de terminaison d'API: <https://wd852cn056.execute-api.eu-north-1.amazonaws.com/dev/step>

▼ Détails

Autorisation: **NONE**
Chemin de ressource: **/step**
Étape: **dev**
ID de déclaration: **59d58115-302d-5260-a7c1-ee8887a81909**
isComplexStatement: **Non**
Méthode: **POST**
Principal du service: **apigateway.amazonaws.com**
Type d'API: **REST**

FIGURE 3 – Configuration de l'API pour la méthode POST

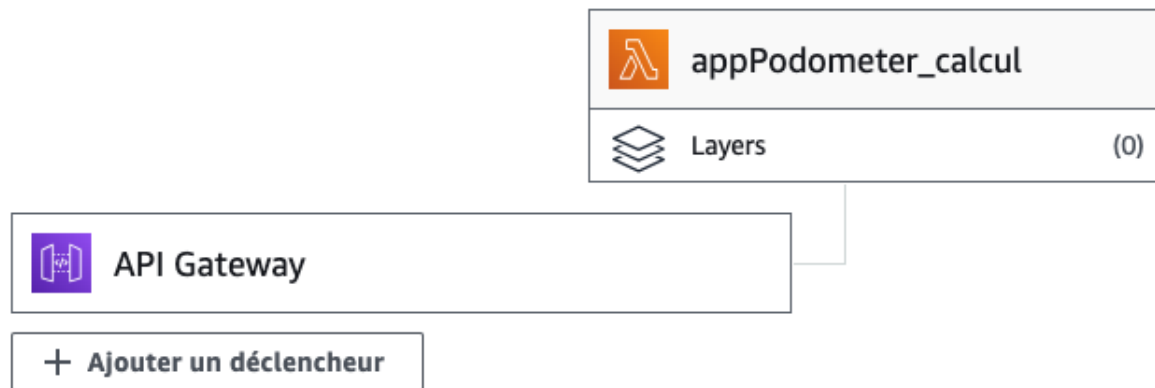


FIGURE 4 – Architecture de l'API de AWS

Voici donc la méthode principale qui s'occupe de gérer la méthode POST et de renvoyer le nombre de pas :

4 Code de Lambda

```

1 // Importation des fonctions fft et util de la biblioth que fft-js
2 const fft = require('fft-js').fft;
3 const fftUtil = require('fft-js').util;
4
5 // D finition de la fonction handler asynchrone export e , qui sera ex cut e lorsque l'
   vnement est re u
6 exports.handler = async (event) => {
7   let steps = 1; // Initialisation du compteur
8   try {

```

```
9 // Log de l' événement reçu
10 console.log('1. Event received:', event);
11
12 // Parse le corps de l' événement pour obtenir les données
13 const body = JSON.parse(event.body);
14 console.log('2. Parsed body:', body);
15
16 // Extraction des données d'accélération verticale du corps
17 const accelerationData = body.verticalAccelerations;
18 console.log('3. Acceleration data:', accelerationData);
19
20 // Vérifie que les données d'accélération sont un tableau
21 if (!Array.isArray(accelerationData)) {
22   throw new Error('Invalid acceleration data'); // Lance une erreur si les données ne
   sont pas un tableau
23 }
24 console.log('4. Valid array');
25
26 // Convertit les données d'accélération en nombres flottants
27 const accelerationArray = accelerationData.map(parseFloat);
28
29 // Suppression de la composante continue (DC) en soustrayant la moyenne de chaque valeur
30 const mean = accelerationArray.reduce((acc, val) => acc + val, 0) / accelerationArray.
   length;
31 const centeredAccelerationData = accelerationArray.map(val => val - mean);
32
33 // Application de la FFT aux données d'accélération centrées
34 const phasors = fft(centeredAccelerationData);
35 console.log('5. Phasors:', phasors);
36
37 // Calcul des magnitudes des phasors
38 const magnitudes = phasors.map(ph => Math.sqrt(ph[0] ** 2 + ph[1] ** 2));
39 console.log('6. Magnitudes:', magnitudes);
40
41 // Détermination de la fréquence d'échantillonnage et de la taille de la FFT
   partir du corps
42 const sampleRate = 50; // 50Hz
43 const fftSize = 750; // mesures --> 15sec de mes
44
45 // Définition des fréquences minimale et maximale d'intérêt (en Hz)
46 const minFrequency = 1; // Fréquence minimale d'intérêt (Hz)
47 const maxFrequency = 3; // Fréquence maximale d'intérêt (Hz)
48
49 // Calcul des indices minimaux et maximaux correspondant aux fréquences d'intérêt
50 const minIndex = Math.ceil(minFrequency * fftSize / sampleRate);
51 const maxIndex = Math.floor(maxFrequency * fftSize / sampleRate);
52
53 // Initialisation des variables pour trouver la magnitude maximale
54 let maxIndexValue = -1;
55 let maxMagnitude = -Infinity;
56
57 // Parcours des magnitudes dans la plage d'indices d'intérêt pour trouver la magnitude
   maximale
58 for (let i = minIndex; i <= maxIndex; i++) {
59   if (magnitudes[i] > maxMagnitude) {
60     maxMagnitude = magnitudes[i];
61     maxIndexValue = i;
62   }
63 }
64
65 // Log de l'indice et de la magnitude maximaux trouvés
66 console.log('Max index:', maxIndexValue);
67 console.log('Max magnitude:', maxMagnitude);
68
```



```

69 // Si une magnitude maximale valide est trouvée, calcul de la fréquence fondamentale
70 if (maxIndexValue !== -1) {
71     const fundamentalFrequency = maxIndexValue * sampleRate / fftSize;
72     console.log('Fundamental frequency:', fundamentalFrequency);
73
74     // Calcul de la cadence (nombre de pas par seconde) à partir de la fréquence
    fondamentale
75     const cadence = fundamentalFrequency; // steps per second
76     const timeElapsed = body.time; // temps en secondes
77     steps = Math.round(cadence * timeElapsed); // Calcul du nombre total de pas
78     console.log('Calculated steps:', steps);
79 }
80
81 // Retourne le nombre de pas calculé avec un code de statut 200
82 return {
83     statusCode: 200,
84     body: JSON.stringify({ steps: steps }),
85 };
86
87 } catch (error) {
88     // Log de l'erreur et retour d'un message d'erreur avec un code de statut 500
89     console.error('Error processing request:', error);
90     return {
91         statusCode: 500,
92         body: JSON.stringify({ error: error.message }),
93     };
94 }
95 };

```

On observe bien dans notre panel la communication entre le client et le serveur sous forme d'un JSON avec des accélérations sous forme d'array.

```

2024-05-24T15:24:21.827Z      18819281-69b3-49a0-8a71-1d81b6c4fcd6      INFO      1. Événement reçu : {
  body: {
    accelerations: [
      1.732, 1.291, 3.116, 3.166, 2.543, 3.278, 4.895, 6.063,
      6.537, 6.436, 6.331, 5.507, 5.117, 5.211, 5.472, 5.426,
      5.045, 4.576, 4.284, 4.246, 4.358, 4.557, 4.763, 4.866,
      4.516, 4.176, 5.11, 6.623, 7.033, 6.525, 5.596, 5.484,
      5.847, 5.797, 5.761, 5.086, 4.416, 4.176, 4.804, 4.093,
      3.76, 4.535, 5.4, 5.211, 4.945, 4.942, 5.28, 4.916,
      4.437, 5.191, 4.6, 4.562, 5.548, 5.852, 5.675, 5.4,
      5.931, 3.956, 4.172, 5.285, 5.546, 5.218, 5.05, 4.729,
      4.557, 4.89, 4.507, 6.092, 4.772, 4.047, 5.208, 5.891,
      6.034, 5.361, 3.446, 3.389, 5.141, 6.056, 5.402, 4.602,
      5.019, 5.328, 5.153, 5.713, 5.105, 4.172, 4.643, 5.467,
      5.97, 6.525, 5.802, 2.747, 2.369, 3.604, 4.464, 4.933,
    ]
  }
}

```

FIGURE 5 – Lecture du panel lors de la transmission des mesures

5 Traitement des Mesures avec une FFT

Pour obtenir une fréquence fondamentale valide, il est crucial de disposer d'un grand nombre de mesures sur une période prolongée. Ainsi, l'observation se fait sur plusieurs périodes.

Principe de la Fonction Lambda

Réception et Parsing de l'Événement

L'événement est reçu, et son corps (`body`) est parsé pour obtenir les données d'accélération (`verticalAccelerations`), la fréquence d'échantillonnage (`samplingFrequency`), la taille de la FFT (`fftSize`) et le temps écoulé (`time`).

5.1 Validation des Données

Il est vérifié que `verticalAccelerations` est bien un tableau. Si ce n'est pas le cas, une erreur est levée.

5.2 Prétraitement des Données

- Les données d'accélération sont converties en nombres flottants.
- La composante continue (DC) est supprimée en soustrayant la moyenne de chaque valeur d'accélération pour centrer les données autour de zéro.

5.3 Calcul de la FFT

La FFT est appliquée aux données d'accélération centrées, produisant une série de phasors (nombres complexes représentant les amplitudes et phases des différentes fréquences composantes).

5.4 Calcul des Magnitudes

Les magnitudes des phasors sont calculées. La magnitude d'un phasor (a, b) est donnée par $\sqrt{a^2 + b^2}$.

5.5 Identification de la Fréquence Fondamentale

- Les indices correspondant aux fréquences d'intérêt (entre 1 Hz et 3 Hz) sont déterminés en fonction de la fréquence d'échantillonnage et de la taille de la FFT.
- La magnitude maximale dans cette plage de fréquences est trouvée. L'indice de cette magnitude correspond à la fréquence fondamentale des mouvements de marche.

5.6 Calcul de la Cadence et des Pas

- La fréquence fondamentale est calculée à partir de l'indice de la magnitude maximale.
- La cadence (nombre de pas par seconde) est déterminée en utilisant cette fréquence fondamentale.
- Le nombre total de pas est calculé en multipliant la cadence par le temps écoulé et en arrondissant le résultat.

5.7 Retour des Résultats

Si tout s'est bien passé, le nombre de pas est renvoyé. En cas d'erreur, un message d'erreur est retourné.

Test Event Name test	
Response { "statusCode": 200, "body": "{\"steps\":4}" }	
Function Logs START RequestId: 3060a83b-56cc-455d-a008-eef6fa2b4bba Version: \$LATEST 2024-05-30T17:01:32.268Z 3060a83b-56cc-455d-a008-eef6fa2b4bba INFO 1. Event received: { body: '{"accelerations": [0.0, 0.31, 0.59, 0.81, 0.95, 1.0, 0.95, 0.81, 0.59, 0.31, 0.0, -0.31, -0.59, -0.81, -0.95, -1.0, -0.95, -0.81, -0.59, -0.31 }]' } 2024-05-30T17:01:32.313Z 3060a83b-56cc-455d-a008-eef6fa2b4bba INFO 2. Parsed body: { accelerations: [0, 0.31, 0.59, 0.81, 0.95, 1, 0.95, 0.81, 0.59, 0.31, 0, -0.31, -0.59, -0.81, -0.95, -1, -0.95, -0.81, -0.59, -0.31, 0, 0.31, 0.59, 0.81, 0.95, 1, 0.95, 0.81, 0.59, 0.31, 0, -0.31, -0.59, -0.81, -0.95, -1, -0.95, -0.81, -0.59, -0.31, 0, 0.31, 0.59, 0.81, 0.95, 1, 0.95, 0.81, 0.59, 0.31, 0, -0.31, -0.59, -0.81, -0.95, -1, -0.95, -0.81, -0.59, -0.31, 0, 0.31, 0.59, 0.81, 0.95, 1, 0.95, 0.81, 0.59, 0.31, 0, -0.31, -0.59, -0.81, -0.95, -1, -0.95, -0.81, -0.59, -0.31, 0,] }	

FIGURE 6 – Phase de Test en direct sur AWS

Avant de connecter notre API à notre serveur, nous avons effectué une phase de test de la fonction lambda. Pour ce faire, nous avons simulé un événement en créant un JSON contenant une liste d'accélération à la bonne fréquence d'échantillonnage, représentant 5 pas. Nous avons ensuite ajusté notre code pour obtenir le nombre de pas attendu.

6 Conclusion

Ce projet nous a permis de réutiliser nos connaissances en développement d'applications mobiles et de nous approprier le service API Gateway d'Amazon. Nous avons appris à communiquer grâce au protocole HTTP et la manipulation des objet JSON.

D'autre part, nous avons du faire appel à nos compétences en analyse de signaux pour trouver la cadence de pas.