



Rapport final

Creation d'une application android servant de riposte
contre le Covid-19

Ferdinand Attivi

Chargé du cours : Mr Aladji

Plan

1 - Introduction

2 – Besoin et Objectifs du projet

2-1 - Contexte

2-2 - Motivation

2-3 – Objectifs

3 – Conception du projet

3-1 – Planification

3-1-1 – Cahier des charges

3-2 – Repartition des taches

3-3 - Repartition des technologies

4 – Réalisation de l’application

4-1 - Stratégie

4-2 - Outils et langages utilisés

4-3 – Présentation de l’application

5 – Conclusion

6 – Perspectives

7 - Bibliographie

Introduction

Grâce à ce mini projet j'ai encore eu l'opportunité de cumuler les connaissances théoriques avec celles de la pratique .

Ceci permet également de rentrer dans la vie active et de découvrir plus précisement le milieu professionnel.

Ce projet consiste à donner une solution numérique qui pourra aider la population à réduire le risque de contamination lié au Covid 19 . Mon choix s'est porté sur une application mobile servant d'informateur pour l'utilisateur .

Dans une première partie je présenterai les besoins et objectifs liés au projet ,comment j'ai géré le projet (mes choix de conception), l'etape de la réalisation et quelques autres points .

L'élaboration de ce rapport a pour principale source mes connaissances acquises tout au long de cette formation scolaire et de mes recherches personnelles

2 – Bésoins et Objectifs du projet

2-1* Contexte

Aujourd’hui , le monde fait fasse a une crise sanitaire lié a une épidémie coronariène du nom de Covid 19 . En tant que développeur il en revient donc de donner une solution numérique pouvant réduire le risque de contamination lié a cet épidemie devenue Pandémie dans certains pays .

Mon choix s'est donc porté sur une application mobile regroupant tous les informations nécessaires liées au nouveau virus .

2-2 * MOTIVATION

Partant de ce postulat, m’ est venu l’idée de trouver une solution pouvant réduire le risque de contamination liée au Covid 19, une crise sanitaire qui sévit partout dans le monde. J’ai ainsi décidé de réaliser une solution numérique pour les populations afin de réduire le risque de contamination lié a cet épidemie, tout en alliant robustesse et stabilité.

C’est avec ces objectifs en tête, que j’ai créé « COVID », la « superbe application ».

2-3 * Objectifs

a. Les objectifs techniques

Étant pour le moment dans un projet universitaire limité en moyens et en temps, j'ai décidé de restreindre mon projet, en sélectionnant les solutions à développer parmi toutes les possibilités permises.

Ainsi, j'ai projeté dans un premier temps de mettre en place un système d'information qui sera le cerveau de l'application. Ajouté à ce cœur de projet, j'entreprends l'étude d'une application mobile adaptée, le choix d'un protocole ouvert et simple à mettre en place .

Cette application a donc pour objectif de donner aux utilisateurs :

- le Nombre de contaminations , de guérison et de décès dans le monde
- les Symptomes de ce virus
- les Précautions à prendre pour ne pas devenir un contaminé

Ainsi l'utilisateur une fois ayant installé l'application pourra connaître un peu plus le virus et saura quoi faire pour ne pas être contaminé

b* Les délais

Le projet a donc débuté et s'achève le jeudi 04 juin 2020, soit un peu plus de 2 semaines. Afin de terminer ce projet ambitieux à temps, il est important de correctement le gérer et de le tenir à jour grâce aux outils de gestion adéquats. Dans cette optique, j'ai utilisé des outils autant présents dans le domaine universitaire que dans le monde du travail : journal de bord,diagramme de Gantt, etc.

Gestion du projet

1 * Planification

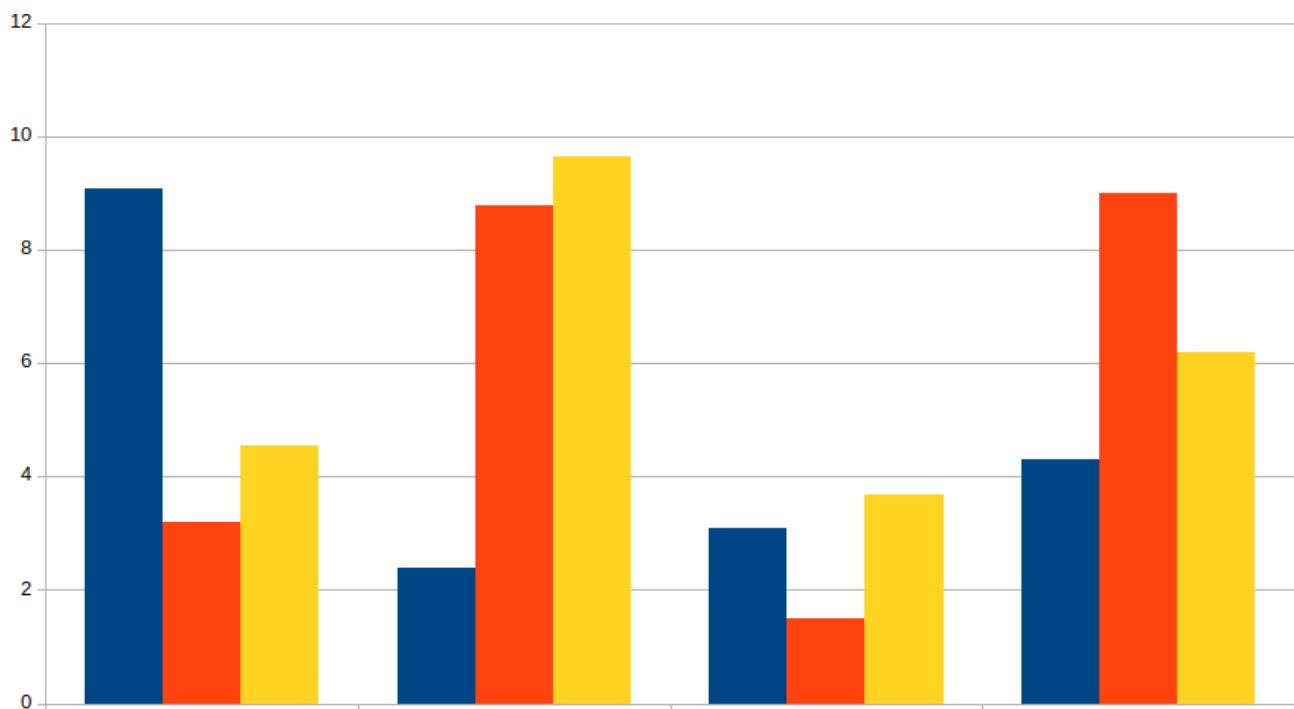
Pour accompagner le développement du projet dès le stade d'ébauche, un cours de projet tuteuré a été mis en place. Ce cours introduit la méthodologie à suivre et les outils nécessaires au bon déroulement d'un projet. Étant dans une dimension ingénieur, cette gestion est d'autant plus importante que le respect des délais, des coûts et de la performance est essentiel dans la conception d'une application mobile. La gestion de projet permet également de créer une base de référence permettant de surveiller les écarts et l'évolution du projet afin d'assurer sa continuité.

3-1-1 * Cahier des charges

Décrivant l'ensemble des conditions attachées à l'exécution du projet, le cahier des charges m'a permis dans un premier temps, de définir le contexte, les enjeux, les objectifs techniques ainsi que les livrables et les axes de développement envisagés. En organisant mes idées, j'ai ainsi pu vérifier la concordance et la faisabilité de ce projet.

3-2 * Répartition des tâches

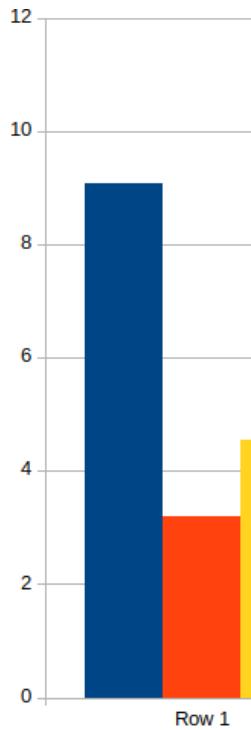
Le diagramme ci-dessous illustre la répartition du temps attribué à chacune des grandes étapes du projet. C'est une conclusion graphique résultant de ma gestion du projet, me permettant d'identifier au premier coup d'œil les travaux chronophages et l'homogénéité entre les tâches.



- | Creation des différents pages de l'application : Layouts (20 %)
- | Creation des différents activités de l'application : Activity (45 %)
- | Creation des différents adaptateurs de l'application : Adapter (35 %)

3-3 * Répartition des technologies

Le diagramme ci-dessous est une visualisation des différentes technologies abordées : le XML et le Kotlin sont les deux langages auxquels j'ai accordé le plus de temps au cours du développement technique.



■ Kotlin

■ XML

* Réalisation du projet

4-1* Stratégie

Ce projet étant personnel, ma première mission fut de définir moi-même une stratégie prévisionnelle ainsi que les objectifs à atteindre. Bien entendu, cette stratégie a évolué au cours du temps afin de satisfaire mes exigences, mais aussi les contraintes auxquelles j'ai pu faire face.

j'ai ainsi tissé ma réflexion, avec pour fil conducteur la relation entre l'utilisateur et le Covid 19

Tout d'abord l'utilisateur : même si celui-ci ne fait pas partie à proprement parler du développement technique, il est primordial d'identifier le public concerné. Quel est le profil de l'utilisateur type ?

Voilà le fondement de ma réflexion. j'ai finalement su que l'application pourrait aider toutes les populations. Ainsi on ne peut plus cibler un public mais tout un monde.

À la manière d'une interface homme-machine, l'application permet d'interagir avec le système, sans nécessiter de connaissances pointues de la part de l'utilisateur pour être configuré et utilisé. Cette application inclut une vue d'ensemble de l'épidémie, les symptômes, les précautions etc

4-2* Outils et langages utilisés

IDE : Android Studio

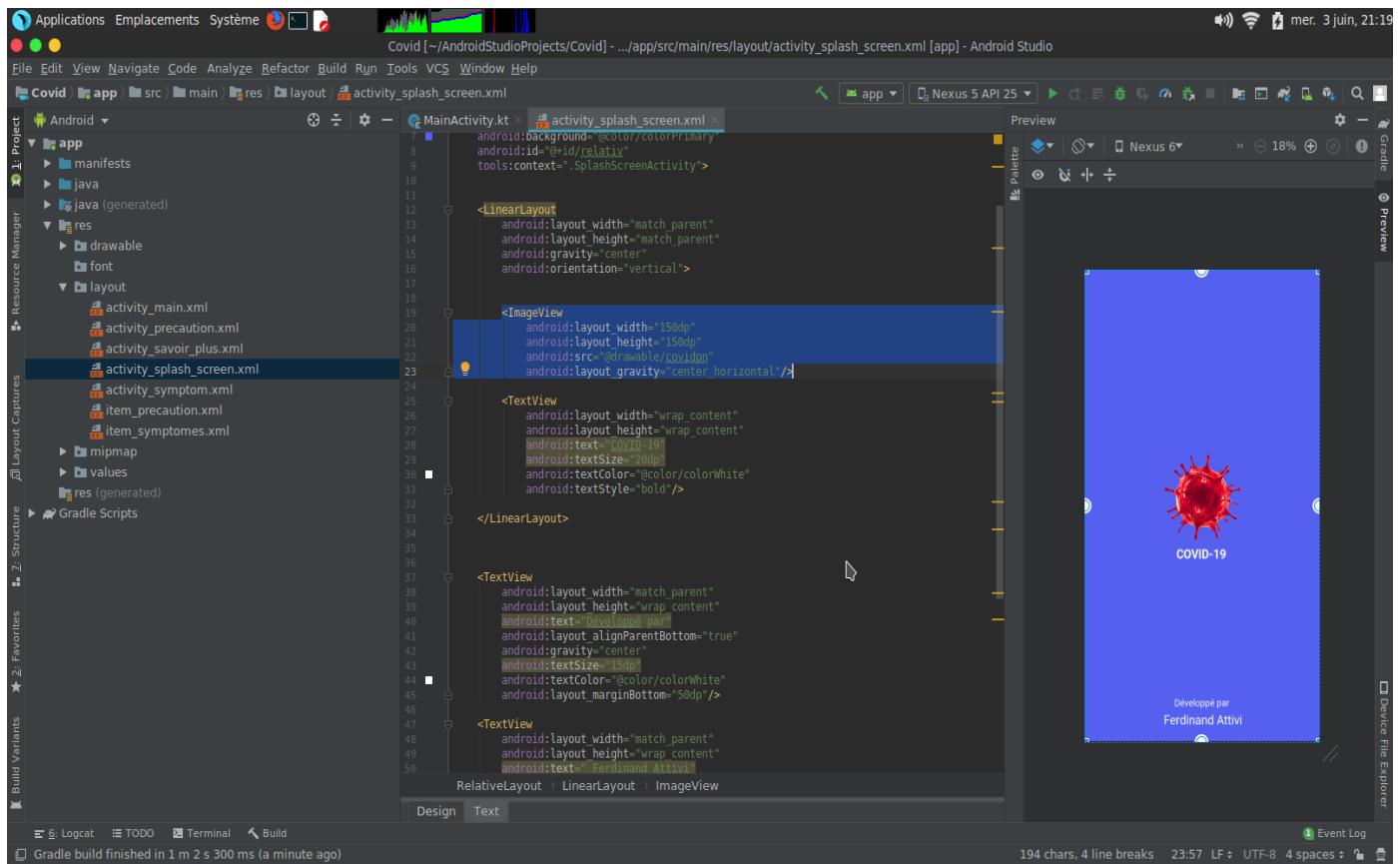
Kotlin : est un langage de programmation orienté objet et fonctionnel, avec un typage statique qui permet de compiler pour la machine virtuelle Java, JavaScript, et vers plusieurs plateformes en natif.

XML : L'Extensible Markup Language, abrégé XML, est un langage de balisage permettant de définir différents espaces de noms, via l'usage de chevrons encadrant les balises.

4-3 – Présentation de l’application

*Ecran d'accueil (SplashScreen)

Première vue de l'utilisateur dans l'application



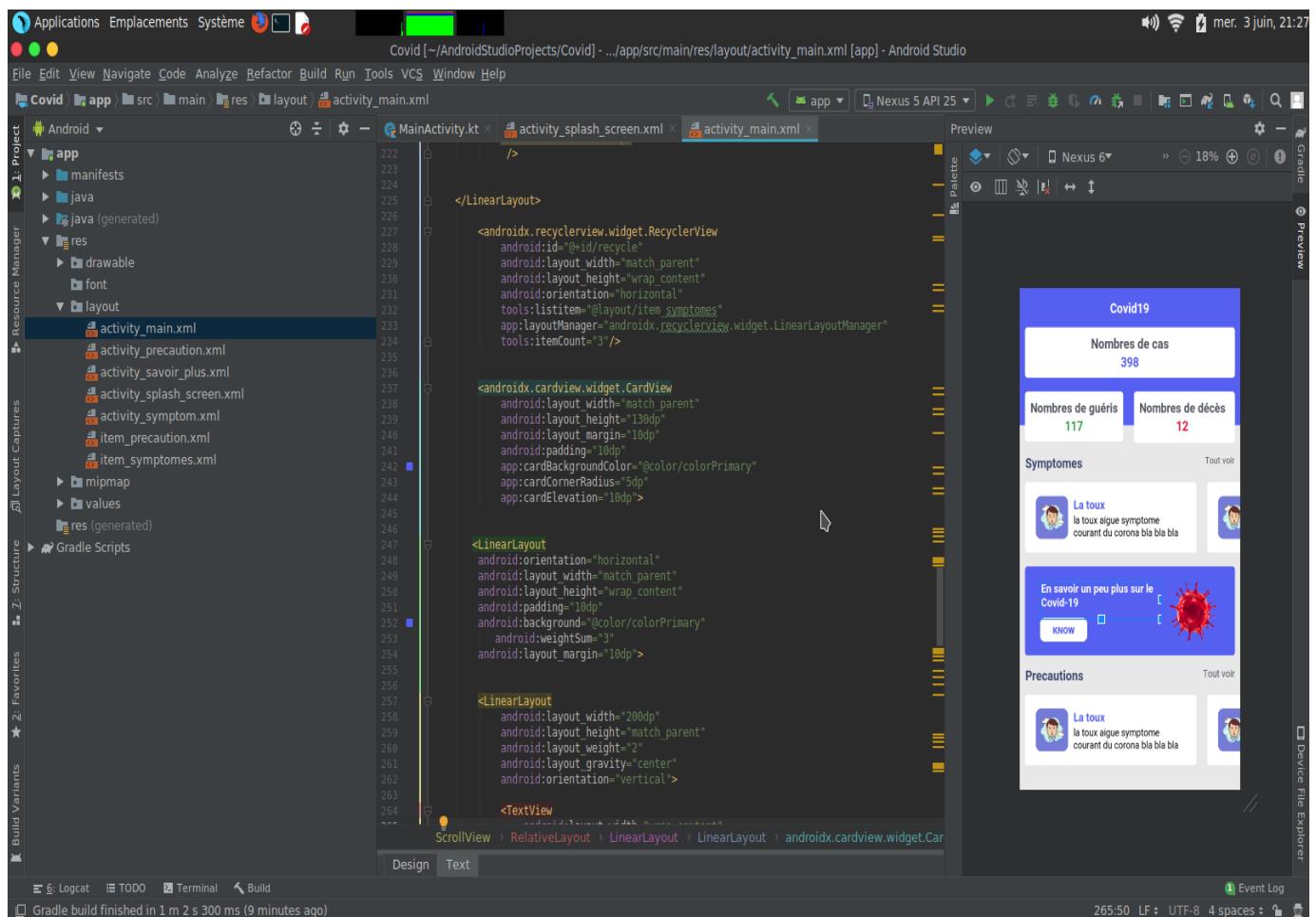
*Menu principal

Page principale contenant les différentes parties de l'application :

- Nombre de cas ,de guérison, de décès.

- Symptomes

- Précautions



*Adapter pour les Symptômes et les précautions

Pour ne pas avoir à créer plus pages il faut donc nécessairement passer par un adapter . Ainsi on peut créer un layout type qu'on multiplier et changer d'éléments à l'affichage



*Creation de la classe Servant de modèle pour les Symptômes

The screenshot shows the Android Studio interface with the following details:

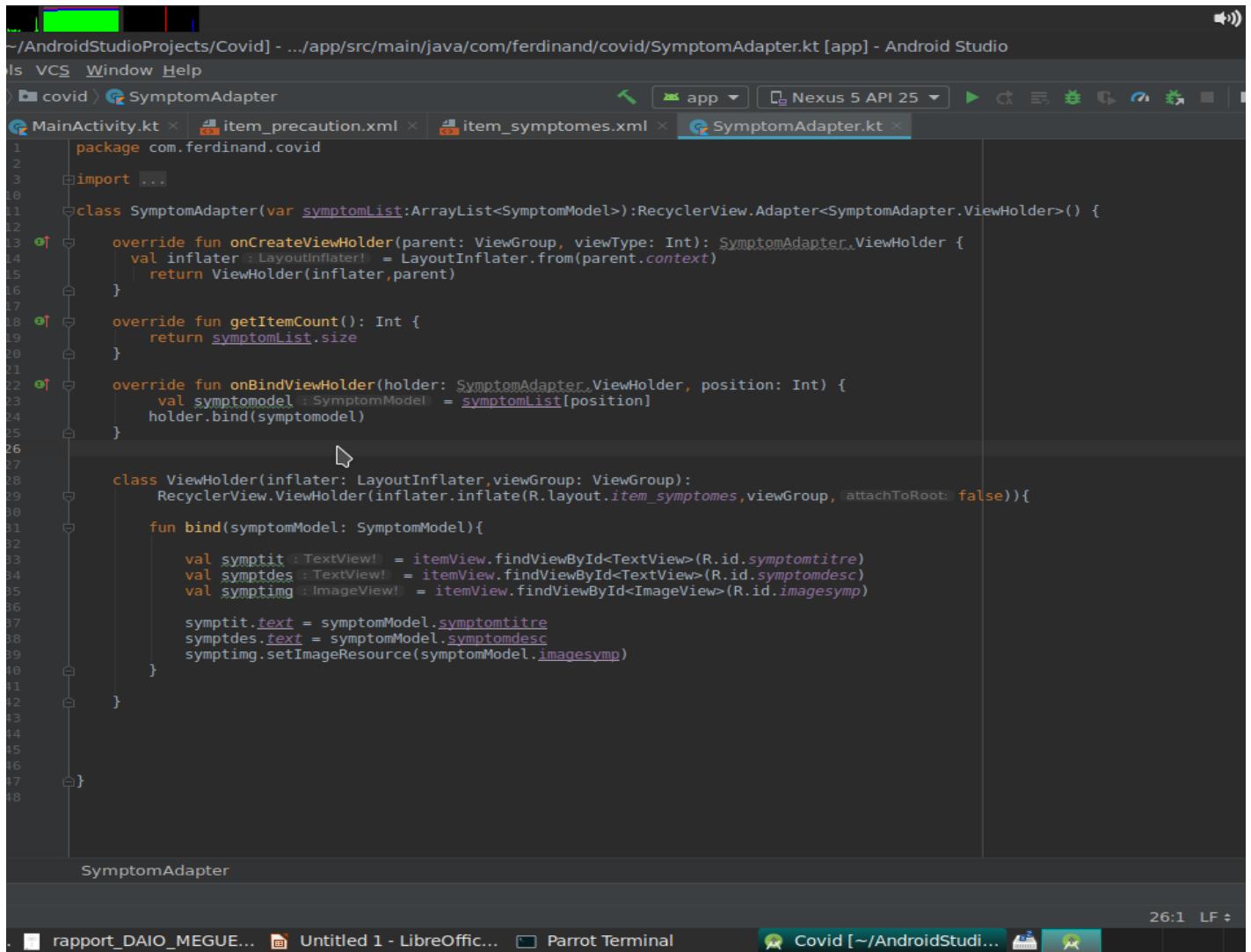
- Project Structure:** The left sidebar shows the project structure for the "Covid" app. It includes the "app" module with Java files like MainActivity.kt, PrecautionActivity.kt, PrecautionAdapter.kt, PrecautionModel.kt, SavoirPlusActivity.kt, SplashScreenActivity.kt, SymptomActivity.kt, and SymptomAdapter.kt. It also includes resources like layout files (activity_main.xml, activity_precaution.xml, activity_savoir_plus.xml, activity_splash_screen.xml, activity_symptom.xml) and XML files (item_precaution.xml, item_symptomes.xml).
- Code Editor:** The main editor window displays the "SymptomModel.kt" file. The code is as follows:

```
package com.ferdinand.covid
class SymptomModel(var imagesym:Int, var symptomtitre:String, var symptomdesc:String) { }
```

The code editor has tabs for MainActivity.kt, item_precaution.xml, item_symptomes.xml, SymptomAdapter.kt, and SymptomModel.kt.

Bottom Bar: The bottom bar shows the status bar with "mer. 3 juin, 21:45", battery level, signal strength, and network connection. It also shows the build status: "Gradle build finished in 1 m 2 s 300 ms (26 minutes ago)". The toolbar includes icons for Logcat, TODO, Terminal, and Build.

*Adapter Symptômes



The screenshot shows the Android Studio interface with the file `SymptomAdapter.kt` open. The code defines a RecyclerView adapter for symptoms. It includes methods for creating view holders, getting item counts, and binding data to view holders. The `ViewHolder` class binds symptom models to views by setting text and image resources.

```
package com.ferdinand.covid

import ...

class SymptomAdapter(var symptomList: ArrayList<SymptomModel>): RecyclerView.Adapter<SymptomAdapter.ViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SymptomAdapter.ViewHolder {
        val inflater: LayoutInflater! = LayoutInflater.from(parent.context)
        return ViewHolder(inflater, parent)
    }

    override fun getItemCount(): Int {
        return symptomList.size
    }

    override fun onBindViewHolder(holder: SymptomAdapter.ViewHolder, position: Int) {
        val symptomodel: SymptomModel = symptomList[position]
        holder.bind(symptomodel)
    }

    class ViewHolder(inflater: LayoutInflater, viewGroup: ViewGroup):
        RecyclerView.ViewHolder(inflater.inflate(R.layout.item_symptomes, viewGroup, false)) {
        fun bind(symptomodel: SymptomModel) {
            val symptit: TextView! = itemView.findViewById<TextView>(R.id.symptomtitre)
            val symptdes: TextView! = itemView.findViewById<TextView>(R.id.symptomdesc)
            val symptimg: ImageView! = itemView.findViewById<ImageView>(R.id.imagesymp)

            symptit.text = symptomodel.symptomtitre
            symptdes.text = symptomodel.symptomdesc
            symptimg.setImageResource(symptomodel.imagesymp)
        }
    }
}
```

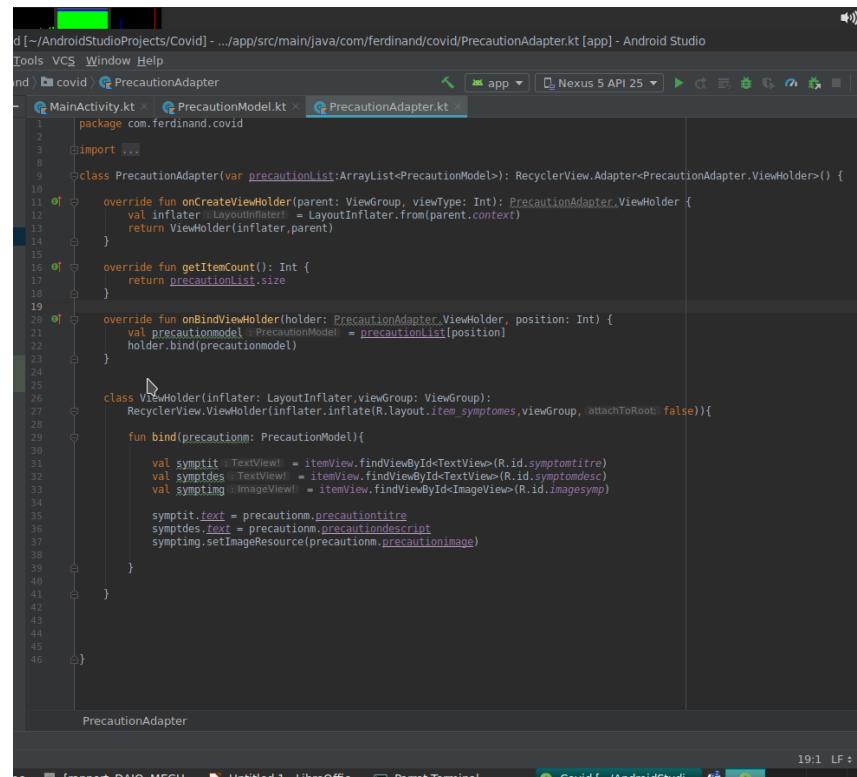
* Creation de la classe Servant de modèle pour les Précautions

The screenshot shows the Android Studio interface with the project 'Covid' open. The code editor displays the 'PrecautionModel.kt' file, which contains the following code:

```
package com.ferdinand.covid
class PrecautionModel(var precautionimage:Int,var precautiontitre:String,var precautiondescript:String)
```

The 'app' module is selected in the Project tool window, showing various Java and XML files. The 'PrecautionModel' class is highlighted in the code editor. The bottom status bar indicates the file is saved (4:2), uses LF line endings, is in UTF-8 encoding, and has 4 spaces.

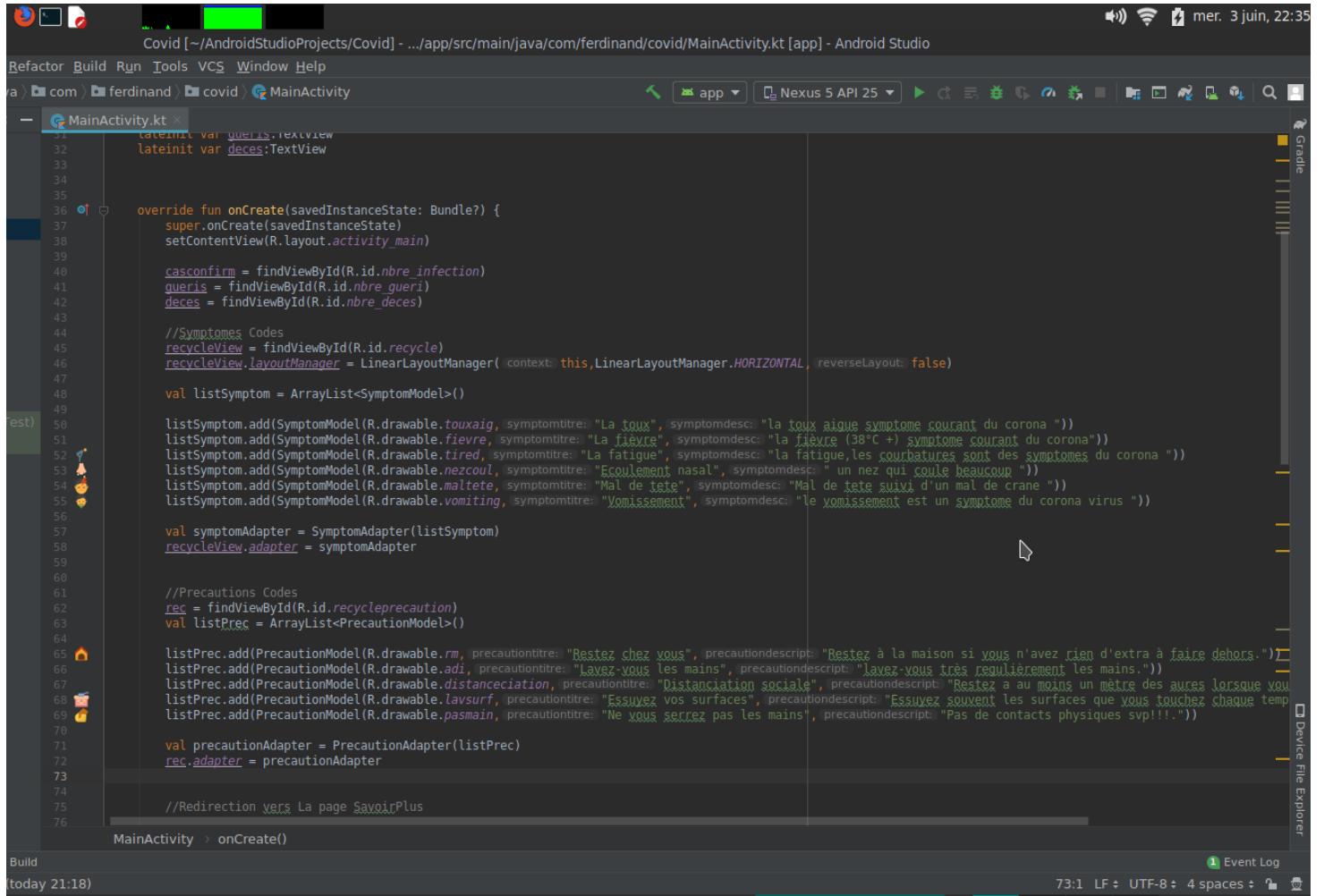
*Adapter Précautions



The screenshot shows the Android Studio interface with the PrecautionAdapter.kt file open in the editor. The code implements a RecyclerView.Adapter for displaying precaution models. It includes methods for creating view holders, getting item counts, and binding data to view holders. The ViewHolders are defined with specific TextViews and an ImageView.

```
1 package com.ferdinand.covid
2
3 import ...
4
5 class PrecautionAdapter(var precautionList:ArrayList<PrecautionModel>): RecyclerView.Adapter<PrecautionAdapter.ViewHolder>() {
6
7     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PrecautionAdapter.ViewHolder {
8         val inflator: LayoutInflater = LayoutInflater.from(parent.context)
9         return ViewHolder(inflator.parent)
10    }
11
12    override fun getItemCount(): Int {
13        return precautionList.size
14    }
15
16    override fun onBindViewHolder(holder: PrecautionAdapter.ViewHolder, position: Int) {
17        val precautionModel: PrecautionModel = precautionList[position]
18        holder.bind(precautionModel)
19    }
20
21    class ViewHolder(inflater: LayoutInflater, itemViewGroup: ViewGroup):
22        RecyclerView.ViewHolder(inflater.inflate(R.layout.item_symptomes, itemViewGroup, attachToRoot: false)) {
23
24        fun bind(precautionm: PrecautionModel) {
25            val symptit: TextView! = itemView.findViewById<TextView>(R.id.symptomtitre)
26            val symptdes: TextView! = itemView.findViewById<TextView>(R.id.symptomdesc)
27            val symptimg: ImageView! = itemView.findViewById<ImageView>(R.id.imagesymp)
28
29            symptit.text = precautionm.precautiontitre
30            symptdes.text = precautionm.precautiondescrit
31            symptimg.setImageResource(precautionm.precautionimage)
32        }
33    }
34
35}
36
37
38
39
40
41
42
43
44
45
46}
```

* Ajout des différents symptômes et précautions



The screenshot shows the Android Studio interface with the file `MainActivity.kt` open. The code is written in Kotlin and handles the `onCreate` method of the main activity. It initializes views for symptoms and precautions, adds them to lists, and sets adapters for RecyclerViews. The code includes comments explaining the purpose of each symptom and precaution.

```

Covid [~/AndroidStudioProjects/Covid] - .../app/src/main/java/com/ferdinand/covid/MainActivity.kt [app] - Android Studio
Refactor Build Run Tools VCS Window Help
com ferdinand covid MainActivity
MainActivity.kt
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    casconfirm = findViewById(R.id.nbre_infection)
    gueris = findViewById(R.id.nbre_gueri)
    deces = findViewById(R.id.nbre_deces)

    //Symptomes Codes
    recycleView = findViewById(R.id.recycle)
    recycleView.layoutManager = LinearLayoutManager(context, LinearLayoutManager.HORIZONTAL, reverseLayout: false)

    val listSymptom = ArrayList<SymptomModel>()

    listSymptom.add(SymptomModel(R.drawable.touxraig, symptomtitre: "La toux", symptomdesc: "la toux aigue symptome courant du corona"))
    listSymptom.add(SymptomModel(R.drawable.fievre, symptomtitre: "La fièvre", symptomdesc: "la fièvre (38°C +) symptome courant du corona"))
    listSymptom.add(SymptomModel(R.drawable.tired, symptomtitre: "La fatigue", symptomdesc: "la fatigue, les courbatures sont des symptomes du corona"))
    listSymptom.add(SymptomModel(R.drawable.nezcoul, symptomtitre: "Ecoulement nasal", symptomdesc: "un nez qui coule beaucoup"))
    listSymptom.add(SymptomModel(R.drawable.malteete, symptomtitre: "Mal de tête", symptomdesc: "Mal de tête suivi d'un mal de crane"))
    listSymptom.add(SymptomModel(R.drawable.vomiting, symptomtitre: "Vomissement", symptomdesc: "le vomissement est un symptome du corona virus"))

    val symptomAdapter = SymptomAdapter(listSymptom)
    recycleView.adapter = symptomAdapter

    //Precautions Codes
    rec = findViewById(R.id.recycleprecaution)
    val listPrec = ArrayList<PrecautionModel>()

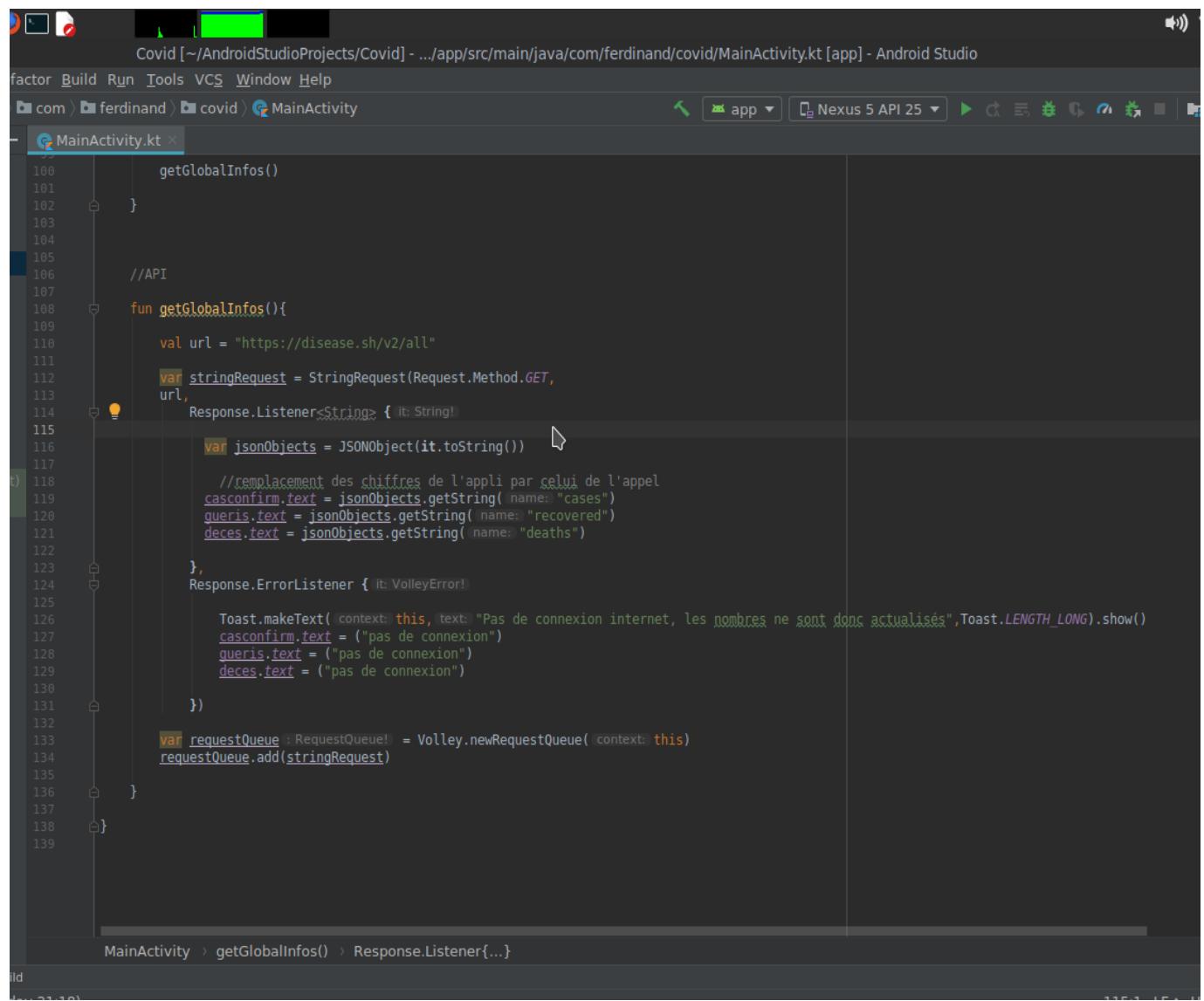
    listPrec.add(PrecautionModel(R.drawable.rm, precautiontitre: "Restez chez vous", precautiondesc: "Restez à la maison si vous n'avez rien d'extra à faire dehors."))
    listPrec.add(PrecautionModel(R.drawable.ad1, precautiontitre: "Lavez-vous les mains", precautiondesc: "Lavez-vous très régulièrement les mains."))
    listPrec.add(PrecautionModel(R.drawable.distanceciation, precautiontitre: "Distanciation sociale", precautiondesc: "Restez à au moins un mètre des autres lorsque vous"))
    listPrec.add(PrecautionModel(R.drawable.lavsurf, precautiontitre: "Essuyez vos surfaces", precautiondesc: "Essuyez souvent les surfaces que vous touchez chaque temp"))
    listPrec.add(PrecautionModel(R.drawable.pasmain, precautiontitre: "Ne vous serrez pas les mains", precautiondesc: "Pas de contacts physiques svp!!!"))

    val precautionAdapter = PrecautionAdapter(listPrec)
    rec.adapter = precautionAdapter

    //Redirection vers La page SavoirPlus
}

```

* Utilisation d'une API pour recevoir les informations

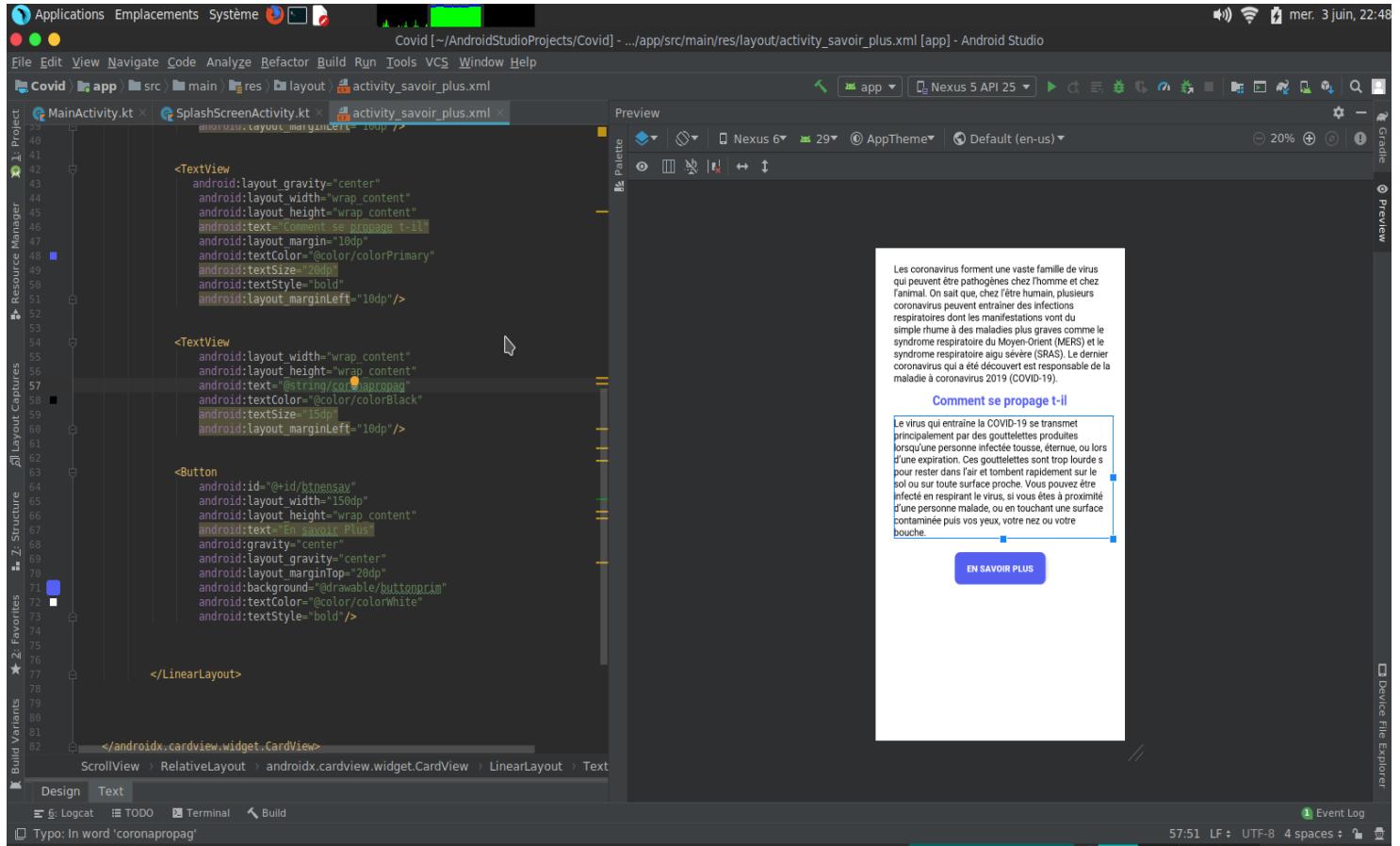


The screenshot shows the Android Studio interface with the project 'Covid' open. The main window displays the 'MainActivity.kt' file. The code implements a function 'getGlobalInfos()' that sends a GET request to the URL 'https://disease.sh/v2/all'. It uses the Volley library to handle the JSON response. If there's no internet connection, it shows a toast message and sets the text of four TextViews ('casconfirm', 'gueris', 'deces') to 'pas de connexion'. Otherwise, it extracts the 'cases', 'recovered', and 'deaths' values from the JSON object and updates the TextViews accordingly.

```
100    getGlobalInfos()
101
102    }
103
104
105    //API
106
107    fun getGlobalInfos(){
108        val url = "https://disease.sh/v2/all"
109
110        var stringRequest = StringRequest(Request.Method.GET,
111            url,
112            Response.Listener<String> { it: String!
113
114                var jsonObjects = JSONObject(it.toString())
115
116                //remplacement des chiffres de l'appli par celui de l'appel
117                casconfirm.text = jsonObjects.getString( name: "cases")
118                gueris.text = jsonObjects.getString( name: "recovered")
119                deces.text = jsonObjects.getString( name: "deaths")
120
121            },
122            Response.ErrorListener { it: VolleyError!
123
124                Toast.makeText( context: this, text: "Pas de connexion internet, les nombres ne sont donc actualisés",Toast.LENGTH_LONG).show()
125                casconfirm.text = ("pas de connexion")
126                gueris.text = ("pas de connexion")
127                deces.text = ("pas de connexion")
128
129            }
130
131        })
132
133        var requestQueue: RequestQueue! = Volley.newRequestQueue( context: this)
134        requestQueue.add(stringRequest)
135
136
137    }
138
139}
```

* Autres activités

Une autre activité pour en savoir un plus sur le virus



Conclusion

Cette année, la crise sanitaire a pris les devants et il en faut donc trouver une solution. De ma réflexion est né un projet pour le moins enrichissant et informateur. Pour la première fois, j'ai mené une étude du papier à la mise en service due l'application. En imaginant ce projet, j'ai en tête quelques bases et méthodes scolaires concernant l'élaboration d'une solution numérique. En maitrisant le projet dans sa globalité, j'ai eu un aperçu complet du processus de développement d'une application mobile, des compétences et des connaissances qui y sont attachées.

D'autre part, Ce projet a cette particularité de rassembler divers corps de métiers : programmation, informatique,Santé et bien d'autres. Cette pluridisciplinarité fut un obstacle que j'ai surmonté grâce aux compétences multiples et ma polyvalence .

Enfin, les temps impartis à la réalisation de l'application furent bref et il a fallu faire preuve de flexibilité et de persévérance, parfois pour respecter les délais, parfois pour respecter les contraintes technologiques imposées par le projet. Somme toute, j'ai retrouvé lors de ces 2 semaines, les compétences, les contraintes mais aussi l'excitation d'un projet d'entreprise.

* Perspectives

Ce projet est avant tout un POC (Proof Of Concept), c'est-à-dire qu'il m'a permis d'affirmer qu'aujourd'hui, il est possible de faire de l'informatique un atout majeur pouvant aider le monde à aller de l'avant et à surmonter les crises.

Il conviendra, cependant, d'élargir le nombre de modules, en gardant à l'esprit que les modules sont passifs et parfois gourmands à stocker.

On pensera par exemple à :

- Ajouter une alerte de notification qui informera l'utilisateur des nouveaux cas ,guérisons ou décès .
- Ajouter une alerte de notification qui notifiera les utilisateurs de se laver les mains
- Ajouter un système d'auto-test qui permettra de se tester et voir si on a le virus ou pas.

Les seules limitations restent l'imagination et les connaissances techniques.

* Bibliographie

Android Studio. [en ligne]

Disponible sur : <https://developer.android.com/>

JSONLint. JSONLint - The JSON Validator. [en ligne]

Disponible sur : <http://jsonlint.com/> (Consulté le 25/04/2020)

KotlinDoc. [en ligne]

Disponible sur : <https://kotlinlang.org/> (Consulté le 02/04/2020)

XML. [en ligne]

Disponible sur : <https://www.w3.org/XML/> (Consulté le 02/04/2020)

Wikipédia. cron - Wikipedia, the free encyclopedia. [en ligne]

Disponible sur : <https://www.wikipedia.org/>