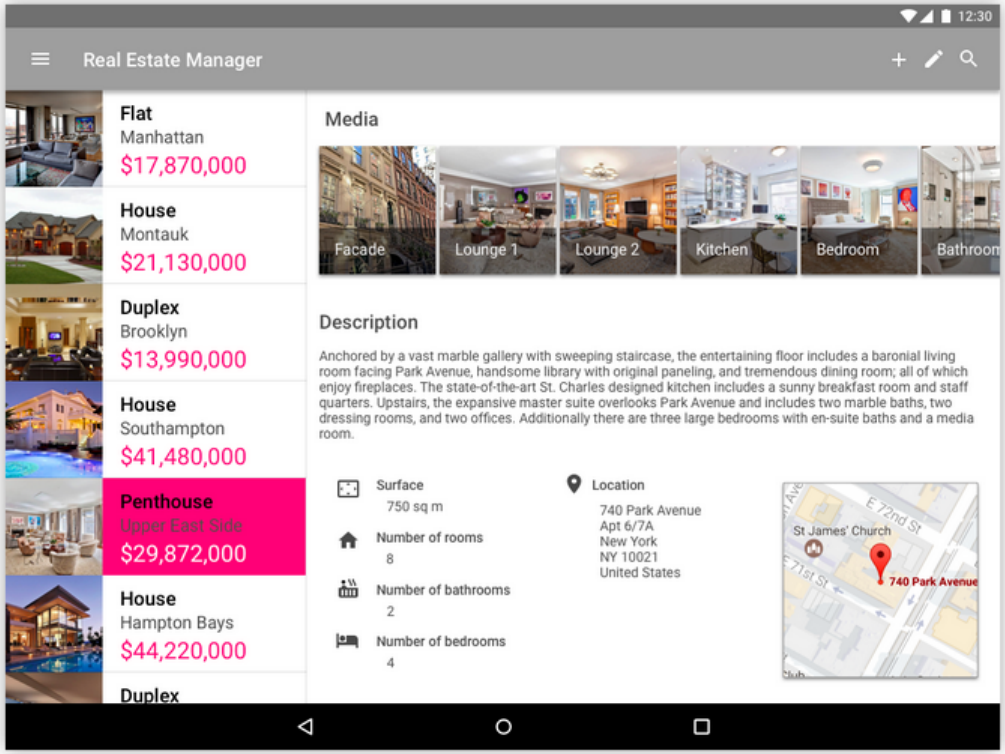
Projet 9 : Devenez un as de la gestion immobilière



La maquette créée par le stagiaire

Projet 9 : Devenez un as de la gestion immobilière

Sommaire :

-Correction des erreurs
-Design de l'application
-Amélioration des fonctionnalités existantes
-Attributs d'un bien immobilier
-Mode hors-ligne
-Gestion des biens immobiliers :
    -Liste des biens
    -Détail d'un bien
    -Ajout et Modification de bien
    -Ajout de photo (La photo peut être récupérée depuis la galerie photos du téléphone, ou prise directement avec l'équipement)
-Géo-localisation
-Moteur de recherche
-Fonctionnalité complémentaire : Intégrer des vidéos dans les informations des biens

Correction des erreurs :

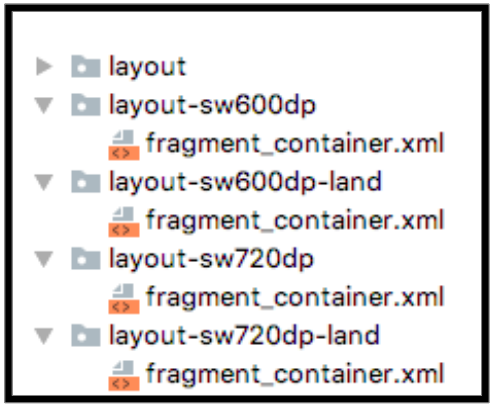
```
public class MainActivity extends AppCompatActivity {

    private TextView textViewMain;
    private TextView textViewQuantity;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        this.textViewMain = findViewById(R.id.activity_second_activity_text_view_main);
        this.textViewQuantity = findViewById(R.id.activity_main_activity_text_view_quantity);

        this.configureTextViewMain();
        this.configureTextViewQuantity();
    }

    private void configureTextViewMain(){
        this.textViewMain.setTextSize(15);
        this.textViewMain.setText("Le premier bien immobilier enregistré vaut ");
    }

    private void configureTextViewQuantity(){
        int quantity = Utils.convertDollarToEuro( dollars: 100);
        this.textViewQuantity.setTextSize(20);
        this.textViewQuantity.setText(quantity);
    }
}
```

Avant

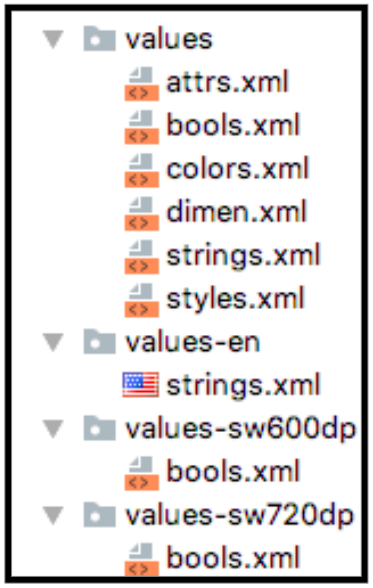
Après

```
public class MainActivity extends AppCompatActivity {

    private TextView textViewMain;
    private TextView textViewQuantity;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        this.textViewMain = findViewById(R.id.activity_main_activity_text_view_main);
        this.textViewQuantity = findViewById(R.id.activity_main_activity_text_view_quantity);

        this.configureTextViewMain();
        this.configureTextViewQuantity();
    }

    private void configureTextViewMain(){
        this.textViewMain.setTextSize(15);
        this.textViewMain.setText("Le premier bien immobilier enregistré vaut ");
    }

    private void configureTextViewQuantity(){
        int quantity = Utils.convertDollarToEuro(100);
        this.textViewQuantity.setTextSize(20);
        this.textViewQuantity.setText(String.valueOf(quantity));
    }
}
```

Design de l'application :




```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.fragment_container)

    mLayout = findViewById(R.id.mlayout)

    tabletSize = resources.getBoolean(R.bool.isTablet)
```

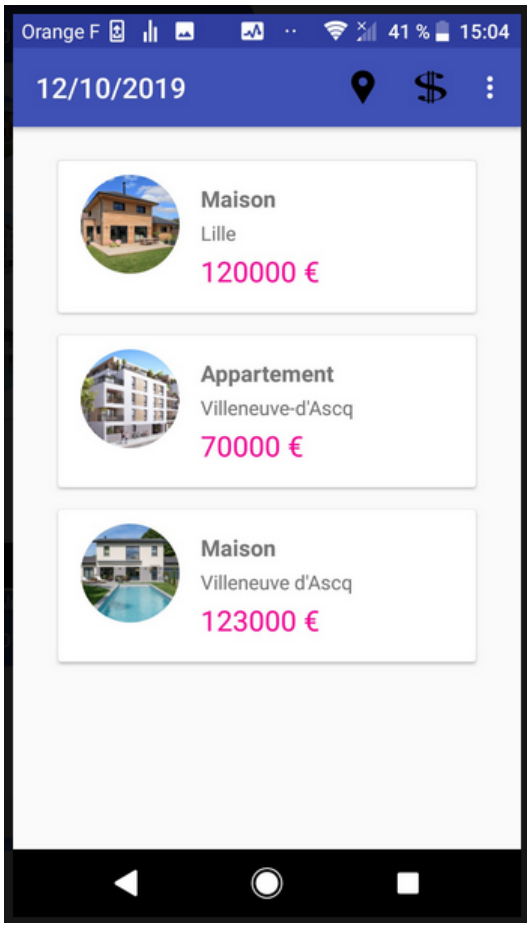


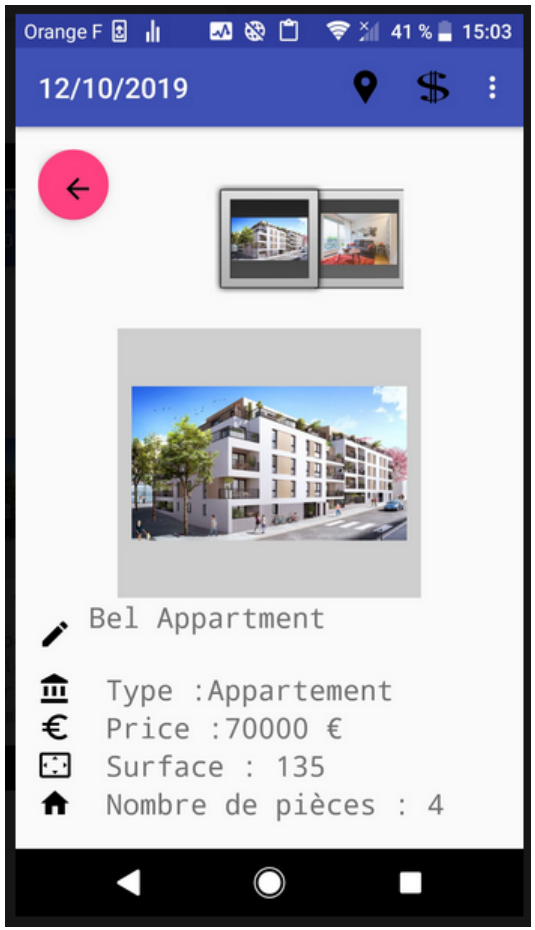

```xml
<resources>
    <bool name="isTablet">false</bool>
</resources>
```

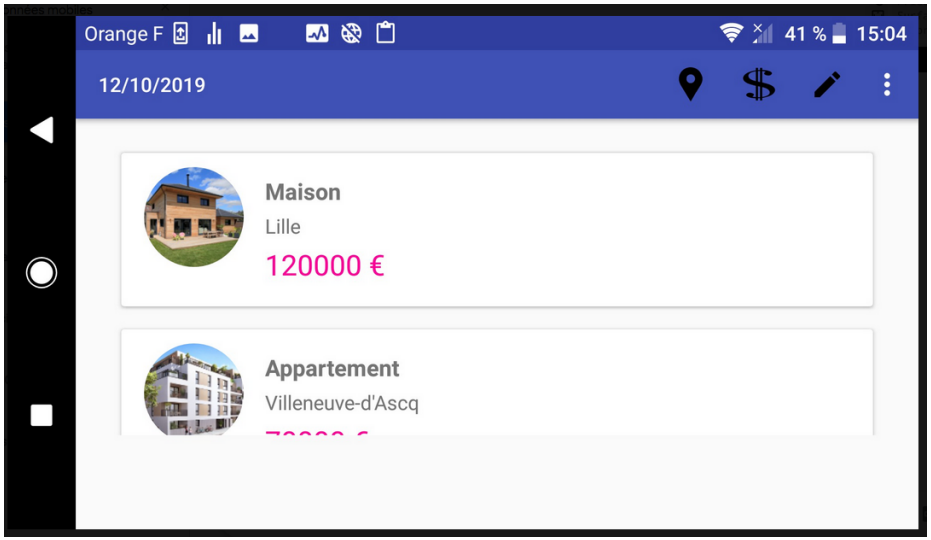
```xml
<resources>
    <bool name="isTablet">true</bool>
</resources>
```

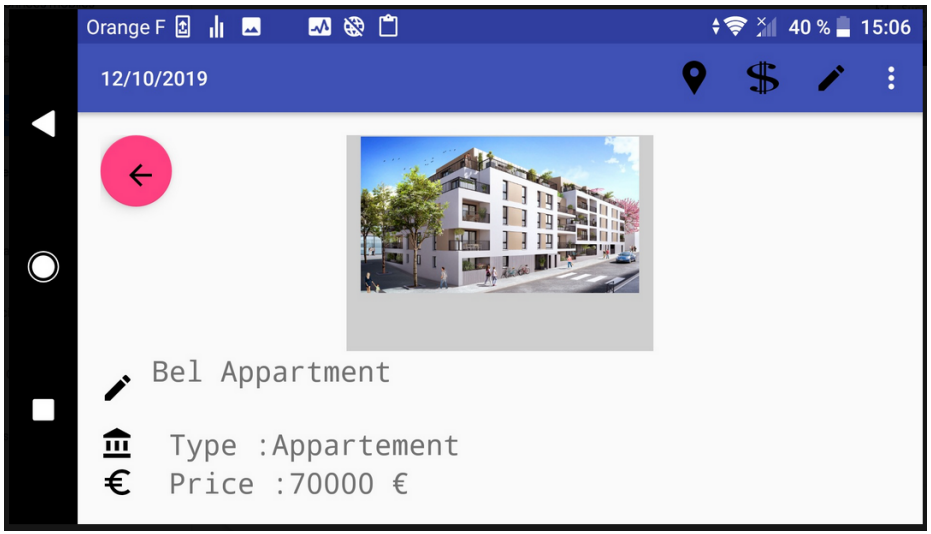
```kotlin
@Subscribe
fun onDetail(event: DetailEvent) {

    val detailFragment = DetailFragment()
    // Supply index input as an argument.
    val args = Bundle()
    args.putParcelable(PROPERTY, event.property)
    args.putSerializable(PROPERTIES, properties)
    args.putSerializable(VIDEOS, videos)
    args.putSerializable(IMAGES, images)
    detailFragment.setArguments(args)

    lastProperty = event.property


    if (tabletSize) {
        fragmentManager.beginTransaction().replace(R.id.content_frame2, detailFragment).commit()
    } else {
        fragmentManager.beginTransaction().replace(R.id.content_frame, detailFragment).commit()
        button.setVisibility(VISIBLE)
    }

}
```

Design de l'application : sur smartphone









Design de l'application : sur tablette

Amélioration des fonctionnalités existantes :

```
val getTodayDate: String
    get() {
        val dateFormat = SimpleDateFormat( pattern: "dd/MM/yyyy")
        return dateFormat.format(Date())
    }
```

```
fun convertDollarToEuro(dollars: Int): Int {
    return Math.round(dollars * 0.812).toInt()
}

fun convertEuroToDollar(euro: Int): Int {
    return Math.round(euro / 0.812 + euro % 0.812).toInt()
}
```

```
@JvmStatic
fun haveInternetConnection(context: Context): Boolean {
    // Fonction haveInternetConnection : return true if connected, return false if not
    val network : NetworkInfo?  = (context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager).activeNetworkInfo

    return !(network == null || !network.isConnected)

}
```

ExampleUnitTest: 3 total, 3 passed

- ExampleUnitTest.convertDollarToEuro_isCorrect
- ExampleUnitTest.returnDate_withGoodFormat
- ExampleUnitTest.convertEuroToDollar_isCorrect

ExampleInstrumentedT...: 2 total, 2 passed

com.openclassrooms.realestatemanager.ExampleInstrumentedTest

- verifyWifi
- useAppContext

Attributs d'un bien immobilier :

```
@Entity
data class Property (@PrimaryKey(autoGenerate = true) var id : Int, var type : String, var price : Int,
                     var nb_bedroom : Int, var nb_bathroom : Int, var surface : Int, var nb_piece : Int,
                     var description : String, var ville : String, var address : String, var proximity : String,
                     var status : String, var start_date: String, var selling_date : String?,
                     var estate_agent : String, var priceIsDollar : String, var nb_photo : Int,
                     var nb_video : Int) : Parcelable {
```

```
@Entity(indices = [Index( ...value: "id_property")],
        foreignKeys = arrayOf(ForeignKey(entity = Property::class,
        parentColumns = arrayOf("id"),
        childColumns = arrayOf("id_property"))))

class Image_property (@PrimaryKey(autoGenerate = true) var id : Int, var id_property: Int
                      , var image : String, var description : String) : Parcelable {
```

```
@Entity(indices = [Index( ...value: "id_property")],
        foreignKeys = arrayOf(ForeignKey(entity = Property::class,
        parentColumns = arrayOf("id"),
        childColumns = arrayOf("id_property"))))
class Video_property (@PrimaryKey(autoGenerate = true) var id : Int,
                      var id_property: Int, var video : String) : Parcelable {
```

Mode hors-ligne : PropertyDao et SaveMyData

```
@Dao
interface PropertyDao {


    @get:Query( value: "SELECT * FROM Property ")
    val properties: LiveData<List<Property>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertProperty(property: Property): Long

    @Query( value: "SELECT * FROM Property WHERE id = :propertyId")
    fun getPropertiesWithCursor(propertyId: Int): Cursor

    @Query( value: "SELECT * FROM Property WHERE type LIKE :type AND
    fun findCorrectProperties(type: String, priceMin: Int, surfaceMi
}
```

```
private fun prepopulateDatabase(): RoomDatabase.Callback {
    return object : RoomDatabase.Callback() {

        override fun onCreate(db: SupportSQLiteDatabase) {

            super.onCreate(db)


            val contentValues = ContentValues()
            contentValues.put("id", 1)
            contentValues.put("type", "Maison")
            contentValues.put("price", 120000)
            contentValues.put("nb_bedroom", 3)
            contentValues.put("nb_bathroom", 1)
            contentValues.put("surface", 135)
            contentValues.put("nb_piece", 4)
            contentValues.put("description", "Belle maison")
            contentValues.put("ville", "Lille")
            contentValues.put("address", "27 Rue Nationale, 59000 Lille")
            contentValues.put("proximity", "métro,école")
            contentValues.put("status", "vendu")
            contentValues.put("start_date", "1999-06-26")
            contentValues.put("selling_date", "1999-06-28")
            contentValues.put("estate_agent", "Denis")
            contentValues.put("priceIsDollar", "Euro")
            contentValues.put("nb_photo", 1)
            contentValues.put("nb_video", 0)
```

```
@Database(entities = [Property::class, Video_property::class, Image_property::class],
abstract class SaveMyData : RoomDatabase() {

    // ~~~ DAO ~~~
    abstract fun propertyDao(): PropertyDao

    abstract fun videoDao(): VideoPropertyDao
    abstract fun imageDao(): ImagePropertyDao

    companion object {

        // ~~~ SINGLETON ~~~
        @Volatile
        private var INSTANCE: SaveMyData? = null


        // ~~~ INSTANCE ~~~
        fun getInstance(context: Context): SaveMyData? {
            if (INSTANCE == null) {
                synchronized(SaveMyData::class.java) {
                    if (INSTANCE == null) {
                        INSTANCE = Room.databaseBuilder(context.applicationContext,
                                SaveMyData::class.java, name: "Database.db")
                                .addCallback(prepopulateDatabase())
                                .build()
                    }
                }
            }
            return INSTANCE
        }
```

Mode hors-ligne : PropertyDataRepository et PropertyViewModel

```
class PropertyDataRepository(private val propertyDao: PropertyDao) {

    val properties: LiveData<List<Property>>
        get() = this.propertyDao.properties

    fun createProperty(property: Property) {
        propertyDao.insertProperty(property)
    }

    fun findCorrectProperties(type: String, priceMin: Int, surfaceMin:
        return propertyDao.findCorrectProperties(type, priceMin, surfa
    }


}
```

```
class PropertyViewModel(// REPOSITORIES
        private val propertyDataSource: PropertyDataRepository, private val

    // DATA
    private var currentProperties: LiveData<List<Property>>? = null


    val properties: LiveData<List<Property>>
        get() = propertyDataSource.properties

    val videos: LiveData<List<Video_property>>
        get() = videoDataSource.videos


    val images: LiveData<List<Image_property>>
        get() = imageDataSource.images

    fun init() {
        if (this.currentProperties != null) {
            return
        }
        currentProperties = propertyDataSource.properties
    }


    fun findCorrectProperties(type: String, priceMin: Int, surfaceMin: Int,
        return propertyDataSource.findCorrectProperties(type, priceMin, sur
    }


    fun createProperty(property: Property) {
        executor.execute { propertyDataSource.createProperty(property) }
    }
```

Mode hors-ligne : ViewModelFactory et Injection

```
class ViewModelFactory(private val propertyDataSource: PropertyDataRepository, private val videoDataSource:

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(PropertyViewModel::class.java)) {
            return PropertyViewModel(propertyDataSource, videoDataSource, imageDataSource, executor) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

```
object Injection {


    fun provideItemDataSource(context: Context): PropertyDataRepository {
        val database :SaveMyData? = SaveMyData.getInstance(context)
        return PropertyDataRepository(database!!.propertyDao())
    }

    fun provideVideoDataSource(context: Context): VideoDataRepository {
        val database :SaveMyData? = SaveMyData.getInstance(context)
        return VideoDataRepository(database!!.videoDao())
    }

    fun provideImageDataSource(context: Context): ImageDataRepository {
        val database :SaveMyData? = SaveMyData.getInstance(context)
        return ImageDataRepository(database!!.imageDao())
    }

    fun provideExecutor(): Executor {
        return Executors.newSingleThreadExecutor()
    }

    fun provideViewModelFactory(context: Context): ViewModelFactory {
        val dataSourceItem :PropertyDataRepository = provideItemDataSource(context)
        val videoSourceItem :VideoDataRepository = provideVideoDataSource(context)
        val imageSourceItem :ImageDataRepository = provideImageDataSource(context)
        val executor :Executor = provideExecutor()
        return ViewModelFactory(dataSourceItem, videoSourceItem, imageSourceItem, executor)
    }
}
```

Mode hors-ligne : PropertyContentProvider

```
class PropertyContentProvider : ContentProvider() {

    override fun onCreate(): Boolean {
        return true
    }

    override fun query(uri: Uri, projection: Array<String>?, selection: String?, selectionArgs: Array<String>?, sortOrder: String?): Cursor? {

        if (context != null) {
            val propertyId :Int  = ContentUris.parseId(uri).toInt()
            val cursor :Cursor  = SaveMyData.getInstance(context!!)!!.propertyDao().getPropertiesWithCursor(propertyId)
            cursor.setNotificationUri(context!!.contentResolver, uri)
            return cursor
        }

        throw IllegalArgumentException("Failed to query row for uri $uri")
    }

    override fun getType(uri: Uri): String? {
        return "vnd.android.cursor.item/$AUTHORITY.$TABLE_NAME"
    }

    override fun insert(uri: Uri, contentValues: ContentValues?): Uri? {

        if (context != null) {
            val id :Long  = SaveMyData.getInstance(context!!)!!.propertyDao().insertProperty(Property.fromContentValues(contentValues!!))
            if (id != 0L) {
                context!!.contentResolver.notifyChange(uri,  observer: null)
                return ContentUris.withAppendedId(uri, id)
            }
        }

        throw IllegalArgumentException("Failed to insert row into $uri")
    }

    override fun delete(uri: Uri, s: String?, strings: Array<String>?): Int { return 0 }
    override fun update(uri: Uri, contentValues: ContentValues?, s: String?, strings: Array<String>?): Int { return 0 }

    companion object {

        // FOR DATA
        val AUTHORITY = "com.openclassrooms"
        val TABLE_NAME = Property::class.java.simpleName
        val URI_PROPERTY = Uri.parse( uriString: "content://$AUTHORITY/$TABLE_NAME")
    }
```

Gestion des biens immobiliers :

-Liste des biens : MainActivity

```
configureViewModel()
getProperties()
getImages()

if (Utils.haveInternetConnection( context: this)) {
    getVideos()
}

val args = Bundle()
args.putSerializable(PROPERTIES, properties)
args.putSerializable(IMAGES, images)
listFragment.setArguments(args)

fragmentManager.beginTransaction().replace(R.id.content_frame,
        listFragment).commit()

button = findViewById(R.id.fab) as FloatingActionButton

button.setVisibility(GONE)

button.setOnClickListener(object : View.OnClickListener {
    override fun onClick(view: View) { getProperties() }
})
```

```
private fun configureViewModel() {
    val mViewModelFactory : ViewModelFactory = Injection
            .provideViewModelFactory( context: this)
    this.propertyViewModel = ViewModelProviders
            .of( activity: this, mViewModelFactory)
            .get(PropertyViewModel::class.java)
    this.propertyViewModel!!.init()
}
```

```
private fun getProperties() {
    propertyViewModel!!.properties.observe( owner: this, Observer<List<Property>> {
        listProperty: List<Property> -> updatePropertiesList(listProperty)} )
}


private fun updatePropertiesList(listProperty: List<Property>) {

    properties.clear()
    for (property : Property in listProperty as ArrayList<Property>){
        properties.add(property)
    }
    fragmentManager.beginTransaction().replace(R.id.content_frame,  listFragment).
    listFragment.adapter.updateData(properties)
    lastProperty = properties.get(0);
    button.setVisibility(GONE)
}

private fun getVideos() {
    propertyViewModel!!.videos.observe( owner: this, Observer<List<Video_property>>
}

private fun updateVideosList(listVideos: List<Video_property>) {

    videos.clear()
    for (video : Video_property in listVideos as ArrayList<Video_property>){
        videos.add(video)
    }
}

private fun getImages() {
    propertyViewModel!!.images.observe( owner: this, Observer<List<Image_property>>
}

private fun updateImagesList(listImage: List<Image_property>) {

    images.clear()
    for (image : Image_property in listImage as ArrayList<Image_property>){
        images.add(image)
    }
```

Gestion des biens immobiliers :

-Liste des biens : ListFragment et MyAdapter

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
    myView = inflater.inflate(R.layout.user_recycler, container,  attachToRoot: false)


    val args : Bundle?  = arguments
    val properties : ArrayList<Property>?  = args!!.getSerializable(PROPERTIES) as ArrayList<Property>?
    val images : ArrayList<Image_property>?  = args.getSerializable(IMAGES) as ArrayList<Image_property>?



    val recyclerView : RecyclerView  = myView.findViewById<View>(R.id.list) as RecyclerView
    recyclerView.layoutManager = LinearLayoutManager(activity)
    adapter = MyAdapter(context, properties, images)
    recyclerView.adapter = adapter



    return myView

}
```

```
@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {

    holder.name.setText(properties.get(position).getType());
    holder.descript.setText(properties.get(position).getVille());


    if (properties.get(position).getPriceIsDollar().compareTo("Dollar") == 0) {
        holder.price.setText("$ " + properties.get(position).getPrice());
    }else{
        holder.price.setText(properties.get(position).getPrice() + " €");
    }

    ArrayList<String> listImages = new ArrayList<>();

    if (images != null) {
        for ( Image_property image : images ) {
            if (properties.get(position).getId() ==  image.getId_property() ) {
                listImages.add(image.getImage());
            }
        }
    }
    holder.root.setOnClickListener(new View.OnClickListener() {
        @SuppressLint("ResourceAsColor")
        @Override
        public void onClick(View v) {

            if (previousSelected!=null){
                previousSelected.setBackgroundColor(WHITE);
            }

            holder.root.setBackgroundColor(R.color.colorAccent);

            previousSelected = holder.root;

            Glide.with(holder.avatar.getContext()) RequestManager
                    .load(listImages.get(0)) RequestBuilder<Drawable>
                    .apply(RequestOptions.circleCropTransform()) RequestBuilder<Drawab
                    .into(holder.avatar);


            EventBus.getDefault().post(new DetailEvent(properties.get(position)));

        }
    });
```

Gestion des biens immobiliers :

-Détail d'un bien : MainActivity et DetailFragment

```
@Subscribe
fun onDetail(event: DetailEvent) {

    val detailFragment = DetailFragment()
    // Supply index input as an argument.
    val args = Bundle()
    args.putParcelable(PROPERTY, event.property)
    args.putSerializable(PROPERTIES, properties)
    args.putSerializable(VIDEOS, videos)
    args.putSerializable(IMAGES, images)
    detailFragment.setArguments(args)

    lastProperty = event.property


    if (tabletSize) {
        fragmentManager.beginTransaction().replace(R.id.content_frame2, detailFragment).commit()
    } else {
        fragmentManager.beginTransaction().replace(R.id.content_frame, detailFragment).commit()
        button.setVisibility(VISIBLE)
    }
```

```
var soldate = " "
if (property!!.selling_date!!.compareTo(getString(R.string.date_default2).toNewDateFormat())!= 0) {
    soldate += property.selling_date!!.toFrenchDateFormat()
}

val description : TextView! = myView.findViewById<TextView>(R.id.description)
description.text = " " + property.description + "\n"
val type : TextView! = myView.findViewById<TextView>(R.id.type)
type.text = type.text.toString() + property.type
val money : TextView! = myView.findViewById<TextView>(R.id.money)
val surface : TextView! = myView.findViewById<TextView>(R.id.surface)
surface.text = surface.text.toString() + " " + property.surface
val home : TextView! = myView.findViewById<TextView>(R.id.home)
home.text = home.text.toString() + " " + property.nb_piece
val bed : TextView! = myView.findViewById<TextView>(R.id.bed)
bed.text = bed.text.toString() + " " + property.nb_bedroom
val bath : TextView! = myView.findViewById<TextView>(R.id.bath)
bath.text = bath.text.toString() + " " + property.nb_bathroom
val location : TextView! = myView.findViewById<TextView>(R.id.location)
location.text = location.text.toString() + property.address
val interest : TextView! = myView.findViewById<TextView>(R.id.interest)
interest.text = interest.text.toString() + property.proximity
val start_date : TextView! = myView.findViewById<TextView>(R.id.start_date)
start_date.text = start_date.text.toString() + " " + property.start_date.toFrenchDateFormat()
val status : TextView! = myView.findViewById<TextView>(R.id.status)
status.text = status.text.toString() + " " + property.status + soldate
val agent : TextView! = myView.findViewById<TextView>(R.id.agent)
agent.text = agent.text.toString() + property.estate_agent

val sold : ImageView! = myView.findViewById<ImageView>(R.id.sold)

val comparison : Int = property.status.compareTo("vendu")

if (comparison != 0) {
    sold.visibility = View.GONE
}

if (property.priceIsDollar === "Dollar") {
    money.text = money.text.toString() + "$ " + property.price.toString()
    money.setCompoundDrawablesWithIntrinsicBounds(R.drawable.ic_dollar,  top: 0,  right: 0,  bottom: 0)


} else {
    money.text = money.text.toString() + property.price.toString() + " €"
    money.setCompoundDrawablesWithIntrinsicBounds(R.drawable.ic_euro,  top: 0,  right: 0,  bottom: 0)
}
```

Gestion des biens immobiliers :

-Détail d'un bien : DetailFragment

```
val listImages = ArrayList<String>()
val listDescription = ArrayList<String>()


if (images != null) {
    for (image : Image_property in images) {
        if (property.id ==  image.id_property ) {
            listImages.add(image.image)
            listDescription.add(image.description)
        }
    }
}

val gallery : Gallery = myView.findViewById<View>(R.id.gallery1) as Gallery
gallery.adapter = ImageAdapter(mContext!!, listImages)
val imageView : ImageView = myView.findViewById<View>(R.id.imageAvatar) as ImageView



Glide.with(mContext!!)
        .load(listImages[0])
        .into(imageView)
```

```
gallery.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
    Toast.makeText(mContext, listDescription.get(position),
            Toast.LENGTH_SHORT).show()
    // display the images selected
    Glide.with(mContext!!)
            .load(listImages.get(position))
            .into(imageView)
}

var listVideos = ArrayList<String>()

if (videos != null) {
    for (video : Video_property in videos) {
        if (property.id ==  video.id_property ) {
            listVideos.add(video.video)
        }
    }
}

recyclerView = myView.findViewById<View>(R.id.recyclerView) as RecyclerView
recyclerView.setHasFixedSize(true)
recyclerView.layoutManager = LinearLayoutManager(mContext)
val videoAdapter = VideoAdapter(listVideos)
recyclerView.adapter = videoAdapter

return myView
```

Gestion des biens immobiliers :

-Ajout et Modification de bien : MainActivity et AddFragment

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {

    super.onCreateOptionsMenu(menu)
    val inflater = menuInflater
    inflater.inflate(R.menu.menu, menu)
    return true

}

override fun onOptionsItemSelected(item: MenuItem): Boolean {

    val i1 = item.itemId
    when (i1) {
        R.id.mapItem -> {


  R.id.modifyItem -> {

      val dialogFragment = MyAddFragment()
      dialogFragment.show(fragmentManager, "Sample Fragment")

      val args = Bundle()
      args.putParcelable(CREATEORMODIFY, lastProperty )
      args.putSerializable(VIDEOS, videos)
      args.putSerializable(IMAGES, images)

      dialogFragment.arguments = args

      return true

  }
  R.id.addItem -> {

      val dialogFragment = MyAddFragment()
      dialogFragment.show(fragmentManager, "Sample Fragment")

      return true

  }
```

```
nb_alea = (Math.random() * 100000).toInt()

var startDate :String  = getTodayDate2

if(modify!=null){
    startDate = modify!!.start_date
    ville.text = modify!!.ville
    prix.text = modify!!.price.toString()
    nb_bedroom.text = modify!!.nb_bedroom.toString()
    nb_bathroom.text = modify!!.nb_bathroom.toString()
    surface.text = modify!!.surface.toString()
    nb_piece.text = modify!!.nb_piece.toString()
    adresse.text = modify!!.address
    agent.text = modify!!.estate_agent
    proximity.text = modify!!.proximity
    description.text = modify!!.description


  if  (modify!!.selling_date!!.compareTo(getString(R.string.date_default2)
                .toNewDateFormat()) != 0) {
      date.text = modify!!.selling_date!!.toFrenchDateFormat()
  }

   if(modify!!.priceIsDollar.compareTo("Dollar")==0){
       dialogSpinnerMoney.setSelection(1)
   }

   if(modify!!.status.compareTo("à louer")==0){
       dialogSpinnerStatu.setSelection(1)
   }else if(modify!!.status.compareTo("vendu")==0){
       dialogSpinnerStatu.setSelection(2)
```

Gestion des biens immobiliers :

-Ajout et Modification de bien : AddFragment

```
if(modify!!.type.compareTo("Appartement")==0){
    dialogSpinnerType.setSelection(1)
}else if(modify!!.status.compareTo("Immeuble")==0){
    dialogSpinnerType.setSelection(2)
}else if(modify!!.status.compareTo("Studio")==0){
    dialogSpinnerType.setSelection(3)
}else if(modify!!.status.compareTo("Villa")==0){
    dialogSpinnerType.setSelection(4)
}else if(modify!!.status.compareTo("Autre")==0){
    dialogSpinnerType.setSelection(5)
}

if ( imageList != null) {
    for (image : Image_property in imageList!!) {
        if (modify!!.id == image.id_property) {
            photoList.add(image)
            photoListStart.add(image)
        }
    }
    nb_photo.text = photoList.size.toString()
}
photo.setText(getString(R.string.delete_image))

var youtubeString = ""

for (video : Video_property in videoList!!) {
    if (modify!!.id == video.id_property) {
        youtubeString += video.video + ","
        nb_video += 1
    }
}
if(youtubeString.length>2) {
youtube.text = youtubeString.substring(0, youtubeString.length - 1)
}

nb_alea = modify!!.id

add.setText("Modifier le bien")
```

```
date.setOnClickListener(View.OnClickListener { it: View!
    // calender class's instance and get current date , month and year from calender
    val c : Calendar = Calendar.getInstance()
    val mYear : Int = c.get(Calendar.YEAR) // current year
    val mMonth : Int = c.get(Calendar.MONTH) // current month
    val mDay : Int = c.get(Calendar.DAY_OF_MONTH) // current day
    // date picker dialog
    val  datePickerDialog = DatePickerDialog(mContext!!,
            DatePickerDialog.OnDateSetListener { _, year, monthOfYear, dayOfMonth ->
                // set day of month , month and year value in the edit text
                var day : String = dayOfMonth.toString()
                if(day.length!=2){ day = "0$day" }
                var month : String = (monthOfYear + 1).toString()
                if(month.length!=2){ month = "0" + ( monthOfYear+1)}
                var texte = "$day/$month/$year"
                date.text = texte
            }, mYear, mMonth, mDay)
    datePickerDialog.show()
})

var dateSelling : String =  "02/01/0000"

if (date.text.toString().length>9){
    dateSelling = date.text.toString()
}

add.setOnClickListener { it: View!

    val dollarEuro : String = dialogSpinnerMoney.getSelectedItem() as String
    val statut : String = dialogSpinnerStatu.getSelectedItem() as String
    val type : String = dialogSpinnerType.getSelectedItem() as String

    val comparison : Int = statut.compareTo("vendu")
```

Gestion des biens immobiliers :

-Ajout et Modification de bien : AddFragment et MainActivity

```
if (prix.text.toString().trim({ it <= ' ' }).isEmpty()) run { prix.setError("Veuil
 else if (photoList.size<1) run { photo.setError("Veuillez ajouter au moins une ph
 else if (nb_bedroom.text.toString().trim({ it <= ' ' }).isEmpty()) run { nb_bedro
 else if (nb_bathroom.text.toString().trim({ it <= ' ' }).isEmpty()) run { nb_bath
 else if (surface.text.toString().trim({ it <= ' ' }).isEmpty()) run { surface.set
 else if (nb_piece.text.toString().trim({ it <= ' ' }).isEmpty()) run { nb_piece.s
 else if (ville.text.toString().trim({ it <= ' ' }).isEmpty()) run { ville.setErro
 else if (adresse.text.toString().trim({ it <= ' ' }).isEmpty()) run { adresse.set
 else if (agent.text.toString().trim({ it <= ' ' }).isEmpty()) run { agent.setErro
 else if (proximity.text.toString().trim({ it <= ' ' }).isEmpty()) run { proximity
 else if (description.text.toString().trim({ it <= ' ' }).isEmpty()) run {descript
 else if (comparison == 0 && selling_date.text.toString().trim({ it <= ' ' }).isEm

        run { selling_date.setError("Veuillez renseignez la date de vente quand l
}else if (comparison != 0 && !selling_date.text.toString().trim({ it <= ' ' }).is
        run { selling_date.setError("Il y a une date alors le bien n'est pas en v
}else{

    val youtubeVideos = Vector<String>()

    if (youtube.text.toString().length > 2) {
        val strings :List<String> = youtube.text.toString().split( ...delimiters: ",")
        for (i :Int in strings.indices) {
            youtubeVideos.add((strings[i]))
        }
    }

   var proximityString :String = proximity.text.toString()
   val proximityStringList = ArrayList<String>()
   if (proximity.text.toString().length > 2) {
        val strings :List<String> = proximity.text.toString().split( ...delimiters: ",")
        for (i :Int in strings.indices) {
            proximityStringList.add((strings[i]))
        }
        proximityStringList.sort()
        proximityString = ""
        for (i :String in proximityStringList) {
            proximityString += i + ","                    }

   }

    val property = Property(nb_alea, type, prix.text.toString().toInt(), nb_bedro
    sendEvent(property)
```

```
fun sendEvent ( property : Property) {

    if(modify==null) {
        Toast.makeText(mContext,"Le bien à bien été ajouté", Toast.LENGTH_SHORT).show()
        EventBus.getDefault().post(AddEvent(property))
    }else{
        Toast.makeText(mContext, "Le bien à bien été modifié" , Toast.LENGTH_SHORT).show()
        EventBus.getDefault().post(ModifyEvent(property))
    }
}
```

```
@Subscribe
fun onAdd(event: AddEvent) {

    propertyViewModel!!.createProperty(event.property)
    getProperties()
}

@Subscribe
fun onModify(event: ModifyEvent) {

    for (  property :Property  in properties ){
        if (property.id == event.property.id){

            propertyViewModel!!.createProperty(event.property)
        }
    }
    getProperties()
}
```

Gestion des biens immobiliers :

-Ajout de photo : AddFragment et CameraActivity

```
        if(modify != null){
            if(photo.text.toString().compareTo("Supprimer les photos")!=0){
                for (photog : Image_property in photoListStart) {
                    EventBus.getDefault().post(DeleteImageEvent(photog.id))
                }
                for (photog : Image_property in photoList) {
                    EventBus.getDefault().post(AddImageEvent(photog))
                }
            }
        }else{
            for (photog : Image_property in photoList) {
                EventBus.getDefault().post(AddImageEvent(photog))
            }
        }
         dismiss()

    }
}

photo.setOnClickListener { it: View!
    if(photo.text.toString().compareTo("Supprimer les photos")==0){
        photo.text = "Ajouter une photo"

        photoList.clear()
        nb_photo.text = "Nombre de photo ajouté :" + photoList.size.toString();

    }else {
        val intent = Intent(mContext, CameraActivity::class.java)
        intent.putExtra(ID, nb_alea )
        startActivityForResult(intent, requestCode: 0)
    }

}

return rootView
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_camera)

    val builder = StrictMode.VmPolicy.Builder()
    StrictMode.setVmPolicy(builder.build())

    val i : Intent! = intent

    photoList = ArrayList()
    id_property = i.getIntExtra(ID,  defaultValue: 0)

    takePictureButton = findViewById<View>(R.id.button_image) as Button
    chooseButton = findViewById<View>(R.id.button_choose) as Button
    descrPhoto = findViewById(R.id.description_photo)

    imageView = findViewById<View>(R.id.imageview) as ImageView

    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.CAMERA
        takePictureButton!!.isEnabled = false
        ActivityCompat.requestPermissions( activity: this, arrayOf(Manifest.permissio
    }
}

override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<Strin
    if (requestCode == 0) {
        if (grantResults.size > 0 && grantResults[0] == PackageManager.PERMISSION_
                && grantResults[1] == PackageManager.PERMISSION_GRANTED) {
            takePictureButton!!.isEnabled = true
        }
    }
}


override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == 100) {
        if (resultCode == Activity.RESULT_OK) {

            Glide.with( activity: this) //SHOWING PREVIEW OF IMAGE
                    .load(this.file)
                    .into(this.imageView!!)
        }
    }
```

Gestion des biens immobiliers :

-Ajout de photo : CameraActivity

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == 100) {
        if (resultCode == Activity.RESULT_OK) {

            Glide.with( activity: this) //SHOWING PREVIEW OF IMAGE
                    .load(this.file)
                    .into(this.imageView!!)
        }
    }

    if (requestCode == 200) {
        if (resultCode == Activity.RESULT_OK) {
            this.file = data!!.data
            Glide.with( activity: this) //SHOWING PREVIEW OF IMAGE
                    .load(this.file)
                    .into(this.imageView!!)

        } else { Toast.makeText( context: this, "Vous n'avez pas sélectionné d'image
    }
}

fun takePicture() {
    val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    file = Uri.fromFile(outputMediaFile)
    intent.putExtra(MediaStore.EXTRA_OUTPUT, file)
    startActivityForResult(intent, requestCode: 100)
}

private val outputMediaFile: File?
    get() {

        val mediaStorageDir = File(Environment.getExternalStoragePublicDirectory(
                Environment.DIRECTORY_PICTURES), child: "CameraDemo")

        if (!mediaStorageDir.exists()) { if (!mediaStorageDir.mkdirs()) {
                return null } }
        val timeStamp : String = SimpleDateFormat( pattern: "yyyyMMdd_HHmmss").format(
        return File( pathname: mediaStorageDir.path + File.separator +
                "IMG_" + timeStamp + ".jpg")
    }

fun choosePicture(view: View) {
    val photoPickerIntent = Intent(Intent.ACTION_PICK)
    photoPickerIntent.type = "image/*"
    startActivityForResult(photoPickerIntent, requestCode: 200)
}
```

```
fun choosePicture(view: View) {
    val photoPickerIntent = Intent(Intent.ACTION_PICK)
    photoPickerIntent.type = "image/*"
    startActivityForResult(photoPickerIntent, requestCode: 200)
}

fun finishThis(view: View) {

    val i2 = Intent()

    if (!descrPhoto.text.toString().trim { it <= ' ' }.isEmpty()) {


        photoList.add(Image_property( id: 0, id_property, file!!.toString(),
                descrPhoto.text.toString()))

        i2.putExtra(PHOTO, photoList)
        this.setResult( resultCode: 1, i2)
        this.finish()
    } else {
        descrPhoto.error = "Ajouter une description à la photo !"
    }

}

fun cancelThis(view: View) {

    val i2 = Intent()
    i2.putExtra(PHOTO, ArrayList<Image_property>())
    this.setResult( resultCode: 1, i2)
    this.finish()
}
```

Gestion des biens immobiliers :

-Ajout de photo : AddFragment et MainActivity

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

    if (data != null) {

        if (resultCode == 1) {
            val s: ArrayList<Image_property> = data.getParcelableArrayListExtra(PHOTO)


            for (photo :Image_property in s) {
                photoList.add(photo)
            }

            nb_photo.text = "Nombre de photo ajouté :" + photoList.size.toString();

        }
    }

    super.onActivityResult(requestCode, resultCode, data)
}
```

```
@Subscribe
fun onAddImage(event: AddImageEvent) {

    propertyViewModel!!.createImage(event.image)
    getImages()
}

@Subscribe
fun onDeleteImage(event: DeleteImageEvent) {

    propertyViewModel!!.deleteImage(event.imageId)
    getImages()
}
```

Géo-localisation : MainActivity

```
possibilityToOpenMap()

val lm :LocationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
if (ActivityCompat.checkSelfPermission( context: this, android.Manifest.permission.ACCESS_FINE_LOCATION)
                .checkSelfPermission( context: this, android.Manifest.permission.ACCESS_COARSE_LOCATION)
    return
}

val location :Location? = lm.getLastKnownLocation(LocationManager.GPS_PROVIDER)
if (location != null) {
    longitude = location.getLongitude()
    latitude = location.getLatitude()
    fragmentManager = supportFragmentManager
} else {
    locationManager = this.getSystemService(Context.LOCATION_SERVICE) as LocationManager
    criteria = Criteria()
    bestProvider = locationManager.getBestProvider(criteria, enabledOnly: true).toString()
    Toast.makeText( context: this, "Le chargement de la localisation peut prendre plus ou m...", Toast.Li
    locationManager.requestLocationUpdates(bestProvider, minTime: 1000, minDistance: 0f, listener: this)
}
```

```
verride fun onOptionsItemSelected(item: MenuItem): Boolean {

    val i1 :Int = item.itemId
    when (i1) {
        R.id.mapItem -> {

            if (Utils.haveInternetConnection( context: this)) {

                val args = Bundle()
                args.putDouble(LAT, latitude)
                args.putDouble(LONG, longitude)
                args.putSerializable(PROPERTIES, properties)
                args.putSerializable(VIDEOS, videos)
                args.putSerializable(IMAGES, images)
                firstFragment.setArguments(args)

                if (tabletSize) {
                    fragmentManager.beginTransaction().replace(R.id.content_frame2,
                            firstFragment).commit()
                } else {
                    fragmentManager.beginTransaction().replace(R.id.content_frame,
                            firstFragment).commit()
                    button.setVisibility(VISIBLE)
                }

            }else{
                Toast.makeText( context: this, "Besoin d'une connexion internet pour
                        Toast.LENGTH_SHORT).show()

            }


            return true
```

Géo-localisation : MapFragment

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                         savedInstanceState: Bundle?): View? {
    // Inflate the layout for this com.openclassrooms.realestatemanager.fragment
    mView = inflater.inflate(R.layout.map_layout, container,  attachToRoot: false)

    val args : Bundle?  = arguments
    lat = args!!.getDouble(LAT)
    lng = args.getDouble(LONG)
    properties = args.getSerializable(PROPERTIES) as ArrayList<Property>
    videos = args.getSerializable(VIDEOS) as ArrayList<Video_property>?
    images = args.getSerializable(IMAGES) as ArrayList<Image_property>?


    val button : FloatingActionButton  = mView!!.findViewById<View>(R.id.recentrer) as FloatingActionButton
    button.setOnClickListener(View.OnClickListener {  it: View!
        if (ActivityCompat.checkSelfPermission(mContext!!, android.Manifest.permission.ACCESS_FINE_LOCATION) != P
                        android.Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED)
            return@OnClickListener
        }
        val here = LatLng(lat!!, lng!!)

        mMap!!.moveCamera(CameraUpdateFactory.newLatLngZoom(here, 14f))
    })

    return mView
}


override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    // Obtain the SupportMapFragment and get notified when the map is ready to be used.
    val mapFragment : SupportMapFragment?  = childFragmentManager.findFragmentById(R.id.map) as SupportMapFragment?
    mapFragment!!.getMapAsync(this)
}
```

```
mMap!!.setOnInfoWindowClickListener { marker ->

    val name : String!  = marker.title


    val comp : Int  = name.compareTo("Ici")

    if (comp != 0) {

    val snippet : String!  = marker.snippet
    val separated : Array<String>  = snippet
            .split(":".toRegex())
            .dropLastWhile { it.isEmpty() }.toTypedArray()


    var propertyThis : Property  = properties!![0]



        val detailFragment = DetailFragment()
        // Supply index input as an argument.
        val args = Bundle()

        for ( property : Property  in properties!!) {
            if(property.id == separated[1].toInt()){
                propertyThis = property
            }
        }
        args.putParcelable(PROPERTY, propertyThis)
        args.putSerializable(PROPERTIES, properties)
        args.putSerializable(VIDEOS, videos)
        args.putSerializable(IMAGES, images)
        detailFragment.setArguments(args)


        fragmentManager!!.beginTransaction()
                .replace(R.id.content_frame, detailFragment)
                .commit()
```

Géo-localisation : MapFragment et DetailActivity

```
override fun onConnected(bundle: Bundle?) {
    if (ActivityCompat.checkSelfPermission(mContext!!, android.Manifest.permission.ACCESS_FINE_LOCATION)
                    .checkSelfPermission(mContext!!, android.Manifest.permission.ACCESS_COARSE_LOCATION)
        return
    }

    val here = LatLng(lat!!, lng!!)


    mMap!!.addMarker(MarkerOptions().position(here).title("Ici").icon(BitmapDescriptorFactory.defaultMark

   for ( property :Property  in properties!!) {
       if (Geocoder.isPresent()) {
           try {
               val location :String  = property.address
               val gc = Geocoder(mContext)
               val addresses :(Mutable)List<Address!>!  = gc.getFromLocationName(location,  maxResults: 5) // ge

               val ll = ArrayList<LatLng>(addresses.size) // A list to save the coordinates if they are a
               for (a :Address!  in addresses) {
                   if (a.hasLatitude() && a.hasLongitude()) {
                       ll.add(LatLng(a.latitude, a.longitude))
                   }
               }
               if (ll.size>0) {
                   mMap!!.addMarker(MarkerOptions().position(ll.get(0))
                           .title(property.type)
                           .snippet(property.address + "\n :" + property.id)
                           .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED)))
               }
           } catch (e: IOException) {
               // handle the exception
           }
       }
    }
    mMap!!.moveCamera(CameraUpdateFactory.newLatLngZoom(here, 14f))
```

```
if (Geocoder.isPresent()) {
    try {
        val location :String  = property!!.address
        val gc = Geocoder(mContext)
        val addresses :(Mutable)List<Address!>!  = gc.
                getFromLocationName(location,  maxResults: 5) // g

        val ll = ArrayList<LatLng>(addresses.size) // A list to
        for (a :Address!  in addresses) {
            if (a.hasLatitude() && a.hasLongitude()) {
                ll.add(LatLng(a.latitude, a.longitude))
            }
        }

        if (ll.size>0) {

            val args2 = Bundle()
            args2.putDouble(LAT, ll.get(0).latitude)
            args2.putDouble(LONG, ll.get(0).longitude)
            args2.putSerializable(PROPERTIES, properties)
            args2.putSerializable(VIDEOS, videos)
            args2.putSerializable(IMAGES, images)
            firstFragment.setArguments(args2)
            fragmentManager.beginTransaction()
                    .replace(R.id.content_frame, firstFragment)
                    .commit()
        }
```

Moteur de recherche : MainActivity et ResearchFragment

```
    R.id.researchItem -> {

        val researchFragment = MyResearchFragment()
        researchFragment.show(fragmentManager, tag: "Sample Fragment")

        return true
    }
}
return super.onOptionsItemSelected(item)
```

```
override fun onCreateView(inflater: LayoutInflat
    val rootView : View! = inflater.inflate(R.lay
    dialog!!.setTitle("Rechercher des biens !")


    var _type : String = "%%"
    var _priceMin : String = "0"
    var _surfaceMin : String = "0"
    var _pieceMin : String = "0"
    var _priceMax : String = "99999999"
    var _surfaceMax : String = "99999999"
    var _pieceMax : String = "99999999"
    var _descript : String = "%%"
    var _ville : String = "%%"
    var _address : String = "%%"
    var _proximity : String = "%%"
    var _statu : String = "%%"
    var _startDate : String = "01/01/0000"
    var _sellingDate : String = "01/01/0000"
    var _agent : String = "%%"
    var _isDollar : String = "%%"
    var _photoMin : String = "0"
    var _photoMax : String = "99999999"
    var _videoMin : String = "0"
    var _videoMax : String = "99999999"
```

```
sellingDate.setOnClickListener(View.OnClickListener { it: View!
    // calender class's instance and get current date , month and year from calender
    val c : Calendar = Calendar.getInstance()
    val mYear : Int = c.get(Calendar.YEAR) // current year
    val mMonth : Int = c.get(Calendar.MONTH) // current month
    val mDay : Int = c.get(Calendar.DAY_OF_MONTH) // current day
    // date picker dialog
    val  datePickerDialog = DatePickerDialog(mContext!!,
            DatePickerDialog.OnDateSetListener { _, year, monthOfYear, dayOfMonth ->
                // set day of month , month and year value in the edit text
                var day : String = dayOfMonth.toString()
                if(day.length!=2){ day = "0$day"
                }
                var month : String = (monthOfYear + 1).toString()
                if(month.length!=2){ month = "0" + ( monthOfYear+1)}
                var texte = "$day/$month/$year"
                sellingDate.text = texte
            }, mYear, mMonth, mDay)
    datePickerDialog.show()
})

research.setOnClickListener { it: View!

    var canSearch = true

        if (switchStatut.isChecked()) { _statu = spinnerStatut.getSelectedItem() as String }

        if (switchType.isChecked()) { _type = spinnerType.getSelectedItem() as String }

        if (switchMoney.isChecked()) { _isDollar = spinnerMoney.getSelectedItem() as String }

        if (switchPrice.isChecked()) {

            _priceMin = prixMin.text.toString()
            _priceMax = prixMax.text.toString()

            if (prixMin.text.toString().length<1) run { prixMin
                    .setError("Veuillez remplir correctement le champs"); canSearch = false}
            if (prixMax.text.toString().length<1) run { prixMax
                    .setError("Veuillez remplir correctement le champs"); canSearch = false}
```

Moteur de recherche : ResearchFragment et MainActivity

```
if (switchProximity.isChecked()) {

    var proximityString :String  = proximity.text.toString()
    val proximityStringList = ArrayList<String>()
    if (proximity.text.toString().length > 2) {
        val strings :List<String>  = proximity.text.toString().split( ...delimiters: ",")
        for (i :Int  in strings.indices) {
            proximityStringList.add((strings[i]))
        }
        proximityStringList.sort()
        proximityString = ""
        for (i :String  in proximityStringList) {
            proximityString +=   i + "%"
        }

    }
    _proximity = "%" + proximityString + "%"

    if (proximity.text.toString().length<1) run { proximity.setError("Veuillez rem
```

```
if(canSearch == true) {
    EventBus.getDefault().post(SearchEvent(_type,  _priceMin.toInt(),  _surfaceMin.toInt(),
            _pieceMin.toInt(),  _priceMax.toInt() ,  _surfaceMax.toInt(),  _pieceMax.toInt(),
            _descript,  _ville,  _address,  _proximity,  _statu,  _startDate.toNewDateFormat(),
            _sellingDate.toNewDateFormat(),  _agent,  _isDollar, _photoMin.toInt(),
            _photoMax.toInt(), _videoMin.toInt(), _videoMax.toInt()))
    dismiss()
}
```

```
@Subscribe
fun onSearch(event: SearchEvent) {
    propertyViewModel!!.findCorrectProperties(event.type,event.priceMin, event.surfaceMin, event.pieceMin,
            event.priceMax,  event.surfaceMax,  event.pieceMax, event.descript,  event.ville,
            event.address,  event.proximity, event.statu,  event.startDate, event.sellingDate, event.agent, event.isDollar,
            event.photoMin, event.photoMax, event.videoMin, event.videoMax)
            .observe( owner: this, Observer<List<Property>> {  listProperty: List<Property> -> updatePropertiesList2(listProperty)} )

}


private fun updatePropertiesList2(listProperty: List<Property>) {

    properties.clear()
    for (property :Property  in listProperty as ArrayList<Property>){
        properties.add(property)
    }
    fragmentManager.beginTransaction().replace(R.id.content_frame,  listFragment).commit()
    listFragment.adapter.updateData(properties)
    button.setVisibility(VISIBLE)
}
```

Fonctionnalité complémentaire : Intégrer des vidéos :

AddFragment

```
var youtubeString = ""

for (video :Video_property in videoList!!) {
    if (modify!!.id == video.id_property) {
        youtubeString += video.video + ","
        nb_video += 1
    }
}
if(youtubeString.length>2) {
youtube.text = youtubeString.substring(0, youtubeString.length - 1)
}

nb_alea = modify!!.id
```

AddFragment

```
if (modify != null) {
    for (vidDelete :Video_property in videoList!!) {
        if (vidDelete.id_property == modify!!.id) {
            EventBus.getDefault().post(DeleteVideoEvent(vidDelete))
        }
    }
}

for (vid :String! in youtubeVideos) {
    EventBus.getDefault().post(AddVideoEvent(Video_property( id: 0,nb_alea,vid)))
}
```

MainActivity

```
@Subscribe
fun onAddVideo(event: AddVideoEvent) {

    propertyViewModel!!.createVideo(event.video)
    getVideos()
}

@Subscribe
fun onDeleteVideo(event: DeleteVideoEvent) {

    propertyViewModel!!.deleteVideo(event.video.id)
    getVideos()
}
```

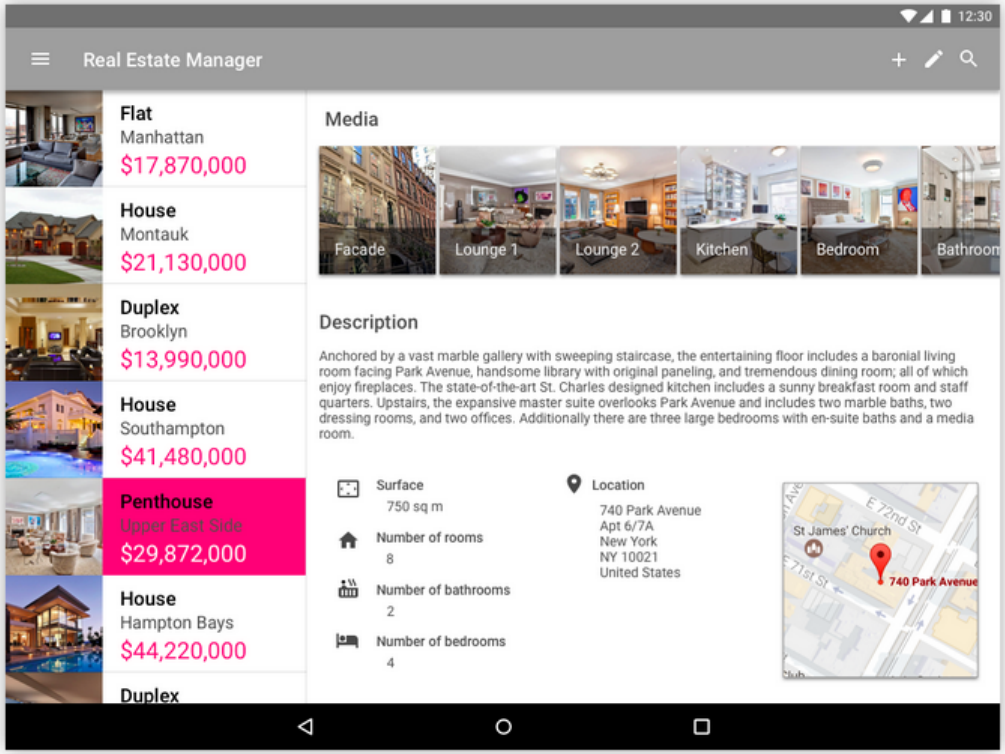
DetailFragment

```
var listVideos = ArrayList<String>()

if (videos != null) {
    for (video :Video_property in videos) {
        if (property.id ==  video.id_property ) {
            listVideos.add(video.video)
        }
    }
}

recyclerView = myView.findViewById<View>(R.id.recyclerView) as RecyclerView
recyclerView.setHasFixedSize(true)
recyclerView.layoutManager = LinearLayoutManager(mContext)
val videoAdapter = VideoAdapter(listVideos)
recyclerView.adapter = videoAdapter
```

Fonctionnalité complémentaire : Intégrer des vidéos : VideoAdapter

```
class VideoAdapter(internal var youtubeVideoList: List<String>) : RecyclerView.Adapter<VideoAdapter.VideoViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): VideoViewHolder {

        val view :View! = LayoutInflater.from(parent.context).inflate(R.layout.video_view, parent, attachToRoot: false)

        return VideoViewHolder(view)

    }

    override fun onBindViewHolder(holder: VideoViewHolder, position: Int) {

        holder.videoWeb.loadData(youtubeVideoList[position].toVideoUrl(), mimeType: "text/html", encoding: "utf-8")

    }


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

    inner class VideoViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

        internal var videoWeb: WebView

        init {

            videoWeb = itemView.findViewById<View>(R.id.videoWebView) as WebView

            videoWeb.settings.javaScriptEnabled = true
            videoWeb.webChromeClient = object : WebChromeClient() {


            }
        }
    }
```

FIN



La maquette créée par le stagiaire