

## Documentation technique WatashiHouse mobile

Liens :

GitHub (code de l'appli) : <https://github.com/DylanGil/WatashiHouse>

APK (lien de l'application à télécharger) :

<https://mega.nz/file/dw4H1ARR#Rn1LuQEmZm1G2H3vIJf5duyezi2e0Ycsug9bSXsiqU>

Contexte :

Fournir une application mobile permettant aux utilisateurs de Watashi house de consulter les articles en ventes, de les acheter, les mettre de côtés, modifier ses informations personnelles...

## Utilisation :

Compte admin :

Email : neo.anderson@gmail.com

Password : nanderson

Compte utilisateur (ou en crée un dans l'appli) :

Email : dga@gmail.com

Password : dga



Lors du démarrage de l'application, un écran de login apparaît demandant à l'utilisateur de s'authentifier ou de créer un compte permettant l'authentification à l'application. Si l'utilisateur s'est déjà connecté, cette page n'apparaîtra pas et passera directement à la page d'accueil.



Email

---

Password

---

SE CONNECTER

Vous n'avez pas encore de compte ? [Cliquer ici](#)

Genre: ☒ Monsieur ☐ Madame

Prenom: Jean

Nom de famille: Dupont

Email: jean.dupont@gmail.com

Mot de passe: 1234

Telephone: 0612345678

Adresse: 2 rue de Paris

Code Postal: 75005

Ville: Paris

Pays: France

Une fois connecté, l'écran d'accueil s'affiche, cet écran montre à l'utilisateur les 8 meilleurs produits. L'utilisateur peut cliquer sur le bouton « Ajouter au panier » pour ajouter l'article au panier. Le bouton étoiles pour ajouter l'article aux favoris. Cliquer sur l'image d'un produit permet de voir ses autres images.

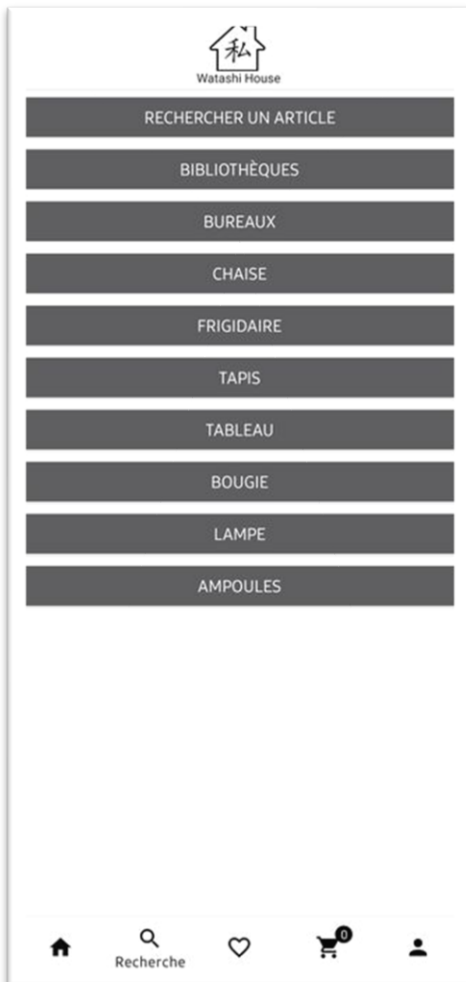


L'utilisateur peut accéder à différentes pages :

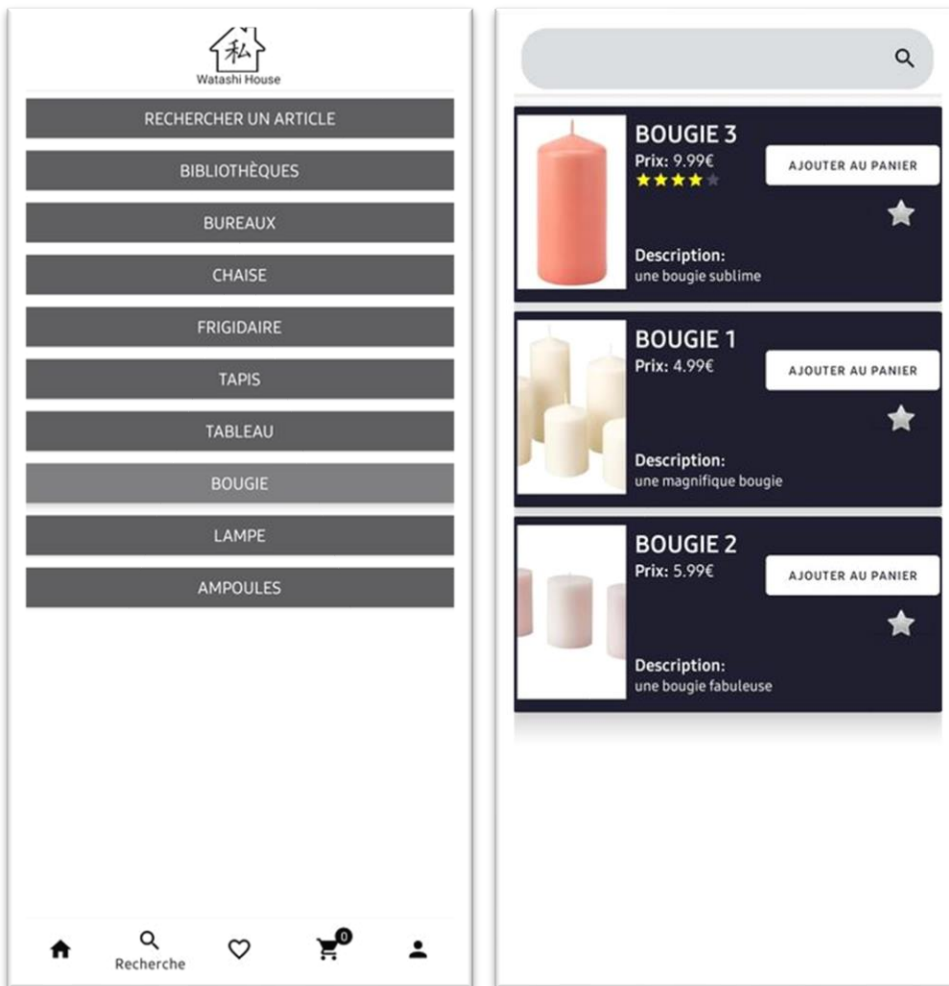
- Recherche

- Favoris
- Panier
- Profil

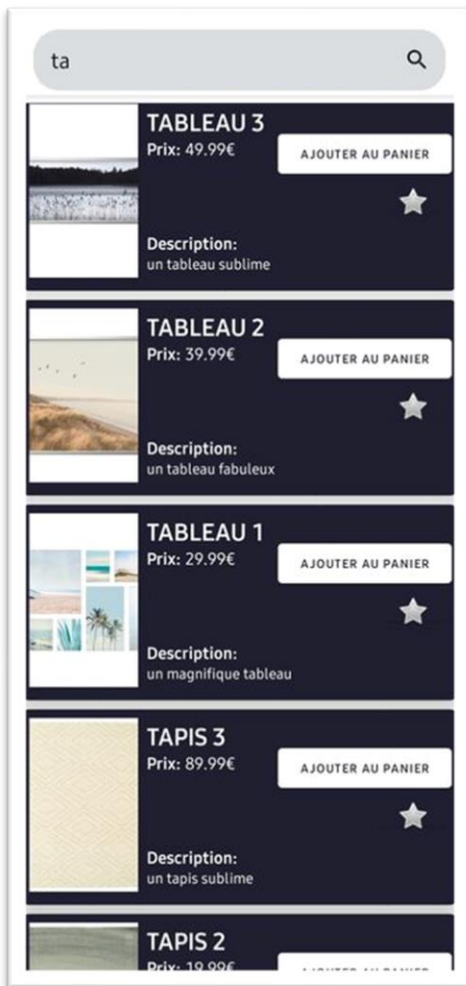
La page recherche permet de rechercher un produit par son nom ou par type d'articles.



Cliquer sur un type de produit permet d'accéder à tous les articles de cette catégorie.



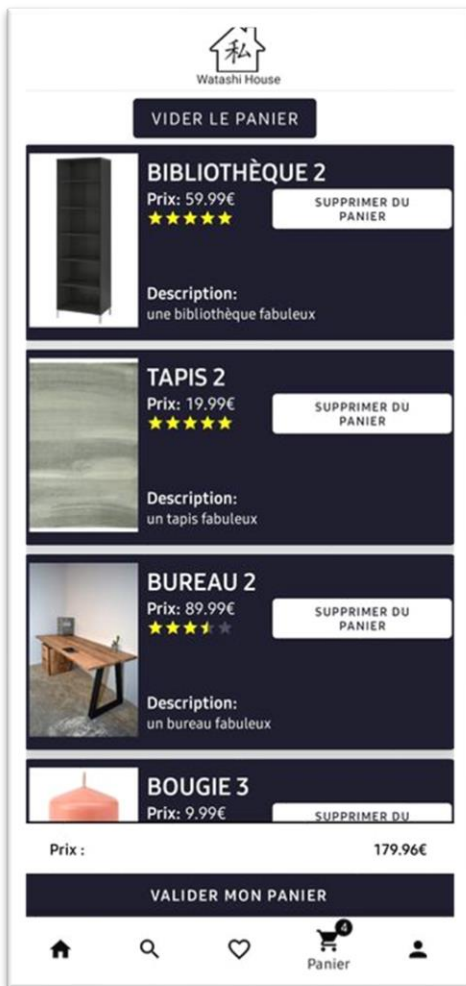
Si l'utilisateur clic sur Rechercher un article, tous les articles apparaissent avec une barre de recherche permettant d'afficher seulement les articles en accord avec ce qui a été écrit dans cette barre de recherche.



La page favoris permet d’afficher tous les articles que l’utilisateur veut garder de côté. L’utilisateur a toujours la possibilité de supprimer un article de ces favoris. La page de favoris peut être vidée avec le bouton “Vider les favoris” présent en haut de la page



La page Panier permet d'afficher les articles que l'utilisateur a mis dans son panier. Un bouton permettant de valider le panier ainsi que le montant du panier sont affichés dans le bas de l'écran. Le bouton pour vider le panier est affiché en haut de la page, de la même manière que dans la page Favoris.




Lors du clic sur Valider le panier, une nouvelle page apparaît avec le montant du panier et différents moyens de méthode (Carte bleu ou GooglePay)



Moyen de paiement :

Buy with card

Buy with  Pay


Prix : 179.96€

Si l'utilisateur décide de payer avec la carte alors une fenêtre lui demandant ces coordonnées bancaires apparaît. Une fois l'achat validé, l'utilisateur est redirigé vers le panier qui a été vidé.

Si l'utilisateur décide de payer avec GooglePay alors il faut qu'il est un compte google avec une carte bleue enregistré dessus. Une fois l'achat validé, l'utilisateur est redirigé vers le panier qui a été vidé.

Moyen de paiement :


Buy with card

Buy with  Pay

× TEST MODE

### Ajoutez vos informations de paiement

Informations concernant la carte bancaire

Numéro de carte bancaire 4242 4242 4242 4242		VISA
MM / AA 12 / 24	CVC 590	

Le numéro de votre carte bancaire n'est pas valide.


Adresse de facturation

Pays  
France

Payer 179,96 €

Moyen de paiement :

Buy with card


Buy with  Pay

Prix : 179,96€

× TEST MODE

### Ajoutez vos informations de paiement

Informations concernant la carte bancaire

Numéro de carte bancaire 4242 4242 4242 4242		VISA
MM / AA 12 / 24	CVC 590	


Adresse de facturation

Pays  
France

Payer 179,96 €

Moyen de paiement :

Buy with card



Buy with  Pay

Prix : 179.96€

✕ TEST MODE


Ajoutez vos informations de paiement

Informations concernant la carte bancaire

Numéro de carte bancaire 4242 4242 4242 4242		
MM / AA 12 / 24	CVC 590	


Adresse de facturation

Pays  
France




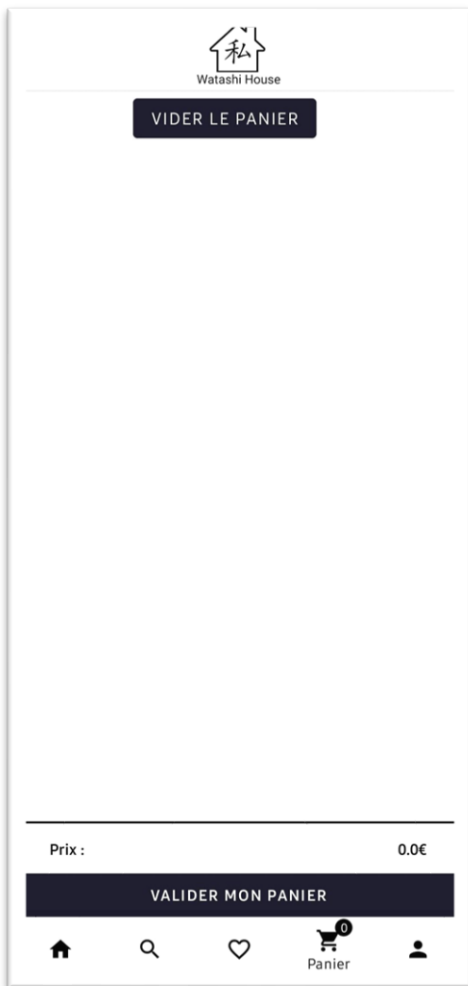
Moyen de paiement :

Buy with card

Buy with  Pay

Prix : 179.96€

 Merci de votre achat



La page Profil permet d'accéder aux informations principales de l'utilisateur. Différents boutons sont présents dans la page :

- Un bouton permettant de modifier toutes les informations de l'utilisateur
- Un bouton permettant de voir l'historique de commande
- Un bouton permettant de se déconnecter



Modifier mes informations :

Mes commandes :

Genre: ☒ Monsieur ☐ Madame

Prenom:

Nom de famille:

Email:

Mot de passe:

Telephone:


Adresse:

Code Postal:

Ville:

Pays:


MODIFIER



N°7881603816

Date d'achat: 2022-06-24


Prix: 29.98€



N°4813555381

Date d'achat: 2022-06-24


Prix: 24.98€



N°7711717178

Date d'achat: 2022-06-24


Prix: 124.97



N°5461687308

Date d'achat: 2022-06-24


Prix: 419.93



N°1632618333

Date d'achat: 2022-06-24


Prix: 284.93



N°4878867454

Date d'achat: 2022-06-26

Prix: 364.92

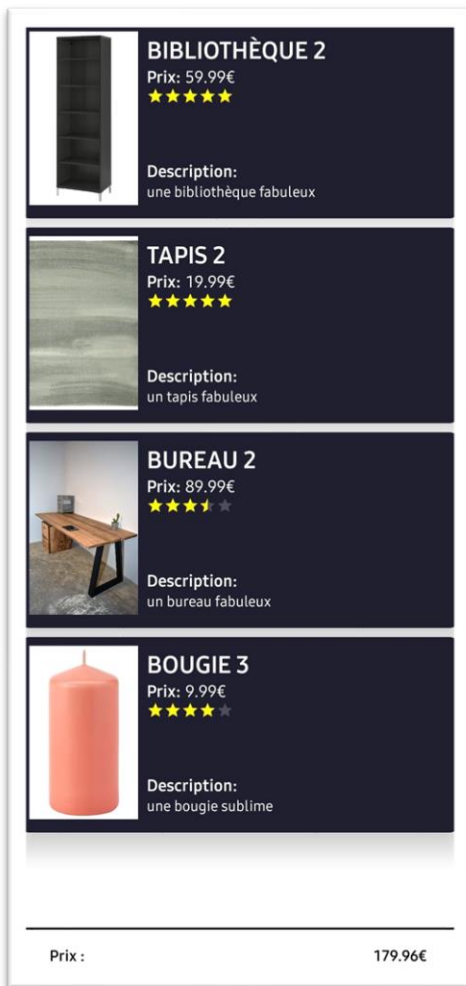


N°1378600170

Date d'achat: 2022-06-26

Prix: 179.96

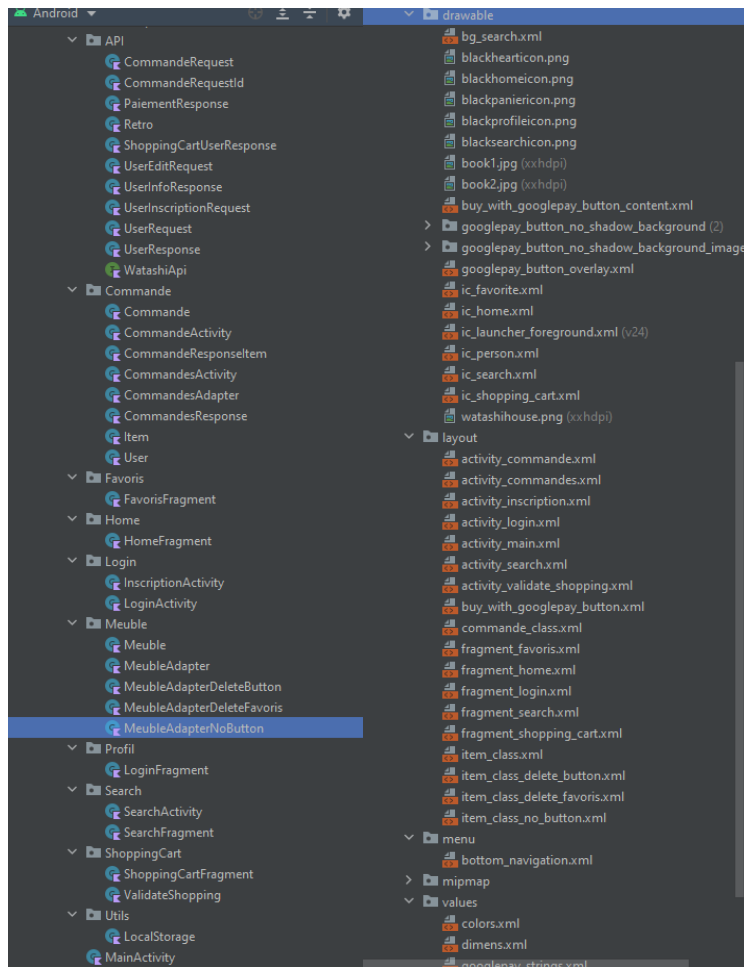
La page mes commandes affichent toutes les commandes effectuées par l'utilisateur. Lors du clic sur l'une des commandes, une nouvelle page apparaît avec tous les articles de la commande ainsi qu'un récap du prix



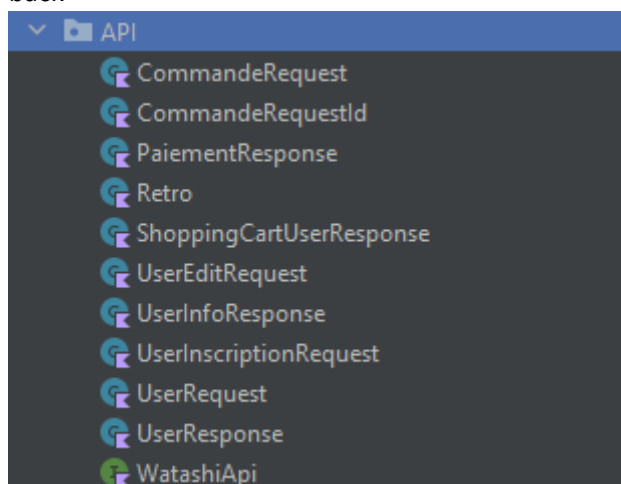
## • Technique

Pour modifier ou consulter le code, il faut utiliser Android Studio ou IntelliJ. Télécharger le code depuis le lien GitHub au début du document. Installer un émulateur android sur cette IDE ou connecté son téléphone à l'émulateur.

Le code est structuré par dossier (nommé comme la page ou il intervienne ou par classe)  
Dans le fichier res on retrouve tout ce qui est graphique (image, affichage de l'écran, icon de l'application...)



Dans la classe API, on retrouve tout ce qui est nécessaire aux appels vers la base de données. Le fichier WatashiAPI stock toutes les routes permettant de GET, POST, DELETE ou PUT. Les fichiers ayant pour fin de nom "Request" sont des classes avec toutes les informations nécessaires à envoyer au back. A l'opposé tout ce qui finissent par "Response" sont les infos reçu du back



WatashiApi.kt



```

interface WatashiApi {
    @POST( value: "utilisateurs/connexion")
    fun login(
        @Body userRequest: UserRequest
    ): Call<UserResponse>

    @POST( value: "utilisateurs/inscription")
    fun register(
        @Body userInscriptionRequest: UserInscriptionRequest
    ): Call<ResponseBody>

    @POST( value: "/commandes")
    fun addToCommande(
        @Body commandeRequest: CommandeRequest,
        @Header( value: "Authentication") Authentication: String): Call<ResponseBody>

    @GET( value: "/commandes/utilisateurs={userId}")
    fun getCommandesFromUser(
        @Path( value: "userId") userId: String,
        @Header( value: "Authentication") Authentication: String): Call<CommandesResponse>

    @GET( value: "/commandes/{commandeId}")
    fun getCommande(
        @Path( value: "commandeId") commandeId: String,
        @Header( value: "Authentication") Authentication: String): Call<JsonObject>
}

```

Pour exécuter ces requêtes, la librairie Retrofit est installé dans l'application.

Login.kt

```

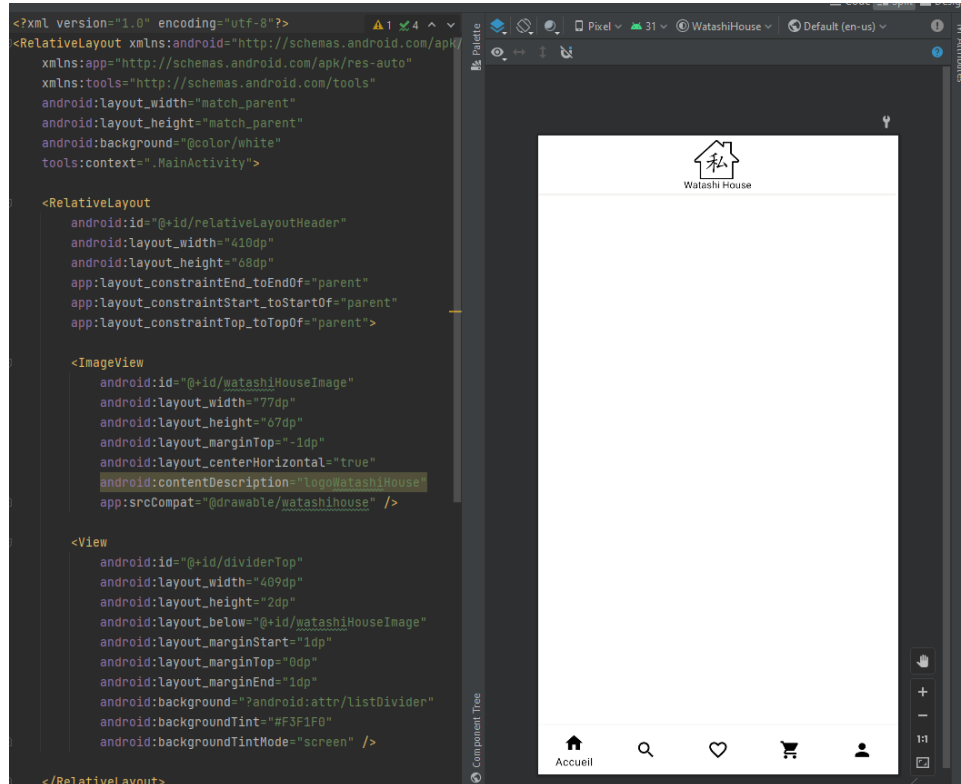
val retrofit = Retrofit().getRetrofitClientInstance().create(WatashiApi::class.java)
retrofit.login(request).enqueue(object : Callback<UserResponse> {
    override fun onResponse(call: Call<UserResponse>, response: Response<UserResponse>) {
        val user = response.body()
        if (user != null) {
            lifecycleScope.launch { this@CoroutineScope
                val localStorage = LocalStorage(applicationContext, datastoreName: "jwt")
                localStorage.clearLocalStorage()
                localStorage.saveToLocalStorage("jwt", user!!.token.toString())
            }
            //passwordTxt.text = Editable.Factory.getInstance().newEditable(user!!.token)
            logOk()
        } else {
            Toast.makeText(applicationContext, text: "User inexistant", Toast.LENGTH_SHORT).show()
        }
        /*Log.e("hash", user!!.data?.hash.toString())
        Log.e("email", user!!.data?.email.toString())*/
    }
})

```

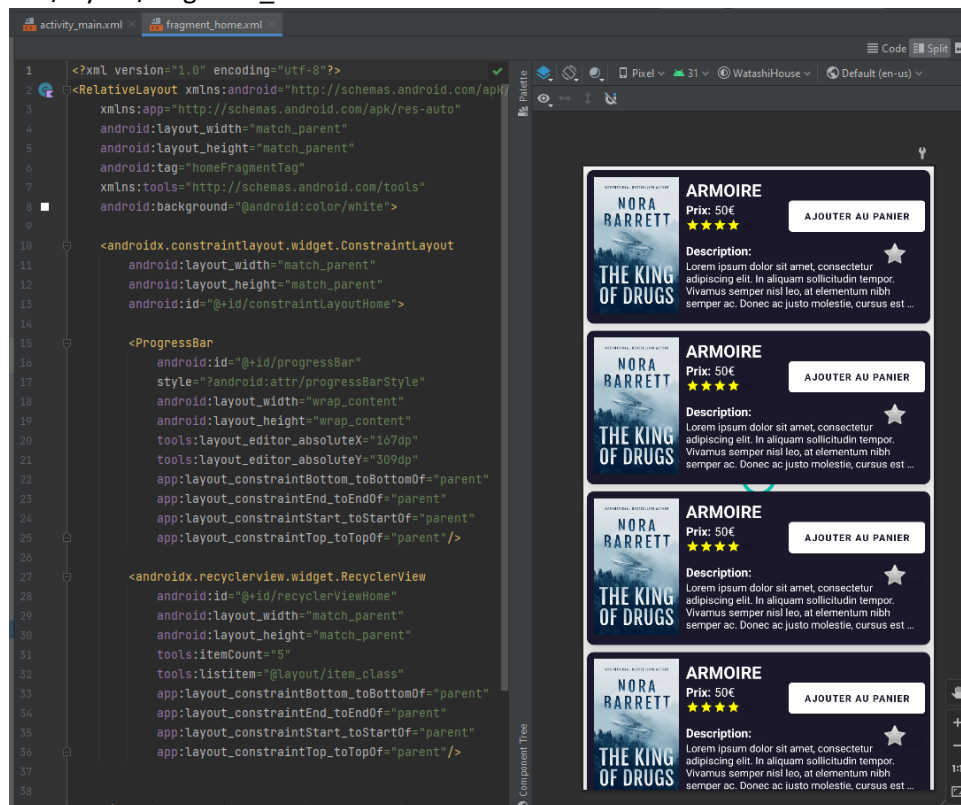
L'application fonctionne grâce des Activités et des fragments. L'application WatashiHouse fonctionne principalement avec une activité et des changements de fragments.

MainActivity.kt est l'activité principale avec l'affichage du logo en haut de la page et la barre de menu en bas de la page. Au milieu de l'activité, ce sont des changements de fragments qui interviennent.

res/layout/activity\_main.xml



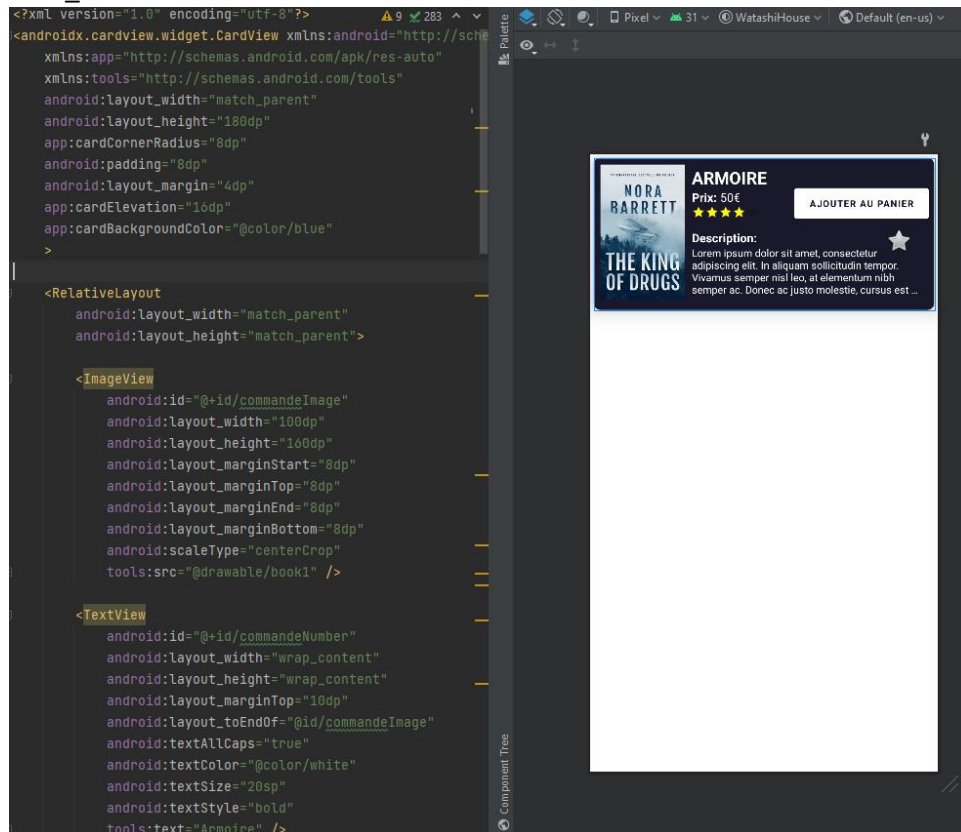
Res/layout/fragment\_home.xml



Par exemple : Le fragment de la page d'accueil s'installera au milieu de l'activité.

Chaque meuble affiché vient d'une classe Meuble

Item\_class.xml



Meuble.kt

```
package com.example.watashihouse.Meuble

data class Meuble (
    var id: String,
    var title: String,
    var summary: String,
    var image1: String,
    var image2: String,
    var image3: String,
    var image4: String,
    var rating: Float,
    var price: String
)
```

MeubleAdapter.kt

```

class MeubleAdapter(var items: List<Meuble>) : RecyclerView.Adapter<MeubleAdapter.MeubleViewHolder>(), Filterable {

    var meubleFilteredList: List<Meuble> = ArrayList()

    init {
        meubleFilteredList = items
    }

    override fun getFilter(): Filter {
        return object: Filter(){
            override fun performFiltering(constraint: CharSequence?): FilterResults {
                var charSearch = constraint.toString()
                if(charSearch.isEmpty()){
                    meubleFilteredList = items
                }else{
                    meubleFilteredList = items.filter { meuble -> meuble.title.lowercase().contains(charSearch.lowercase()) }
                }
                val filterResult = FilterResults()
                filterResult.values = meubleFilteredList
                return filterResult
            }
        }

        override fun publishResults(constraint: CharSequence?, results: FilterResults?) {
            meubleFilteredList = results?.values as ArrayList<Meuble>
            notifyDataSetChanged()
        }
    }
}

```

Chaque item possède un adapter permettant de donner les valeurs par défaut ou les valeurs passé en paramètre `Meuble(id, name, description, imq1,imq2,imq3,imq4, avis, price.toString())` ainsi que définir les actions lors d'un clic.

```

addToFavorite.setOnClickListener { it: View?
    val localStorage = LocalStorage(itemView.context, datastoreName: "jwt")
    val retro = Retro().getRetroClientInstance().create(WatashiApi::class.java)
    retro.addToFavorite(localStorage.favoritesId, meuble.id, localStorage.jwtToken).enqueue(object : Callback<ResponseBd> {
        @SuppressLint(...)value: "RestrictedApi")
        override fun onResponse(call: Call<ResponseBody>, response: Response<ResponseBody>) {
            Toast.makeText(itemView.rootView.context, text: meuble.title + " ajouté au favoris", Toast.LENGTH_SHORT).show()
        }

        override fun onFailure(call: Call<ResponseBody>, t: Throwable) {
            Log.e(tag: "Error", t.message.toString())
            Toast.makeText(itemView.rootView.context, text: "Erreur: Redémarrer l'application", Toast.LENGTH_SHORT).show()
        }
    })
}
}

```

Le fonctionnement pour l'affichage des commandes est similaire avec la classe "Commande".

Dans le dossier "Utils", on peut y retrouver la classe "LocalStorage" qui permet de lire les infos depuis le localStorage du téléphone. Les informations stockées sont les informations de l'utilisateur lors de sa connexion

```

val retro = Retro().getRetroClientInstance().create(WatashiApi::class.java)
retro.login(request).enqueue(object : Callback<UserResponse> {
    override fun onResponse(call: Call<UserResponse>, response: Response<UserResponse>) {
        val user = response.body()
        if (user != null) {
            lifecycleScope.launch { this: CoroutineScope
                val localStorage = LocalStorage(applicationContext, datastoreName: "jwt")
                localStorage.clearLocalStorage()
                localStorage.saveToLocalStorage("jwt", user!!.token.toString())
            }
            //passwordTxt.text = Editable.Factory.getInstance().newEditable(user!!.token)
            logOk()
        } else {
            Toast.makeText(applicationContext, text: "User inexistant", Toast.LENGTH_SHORT).show()
        }
        /*Log.e("hash", user!!.data?.hash.toString())
        Log.e("email", user!!.data?.email.toString())*/
    }
})
}

```

```

var jwtToken = ""
var userId = ""
var userFirstName = ""
var userLastName = ""
var userEmail = ""
var panierId = ""
var favorisId = ""

```

Une fois stocké dans le localStorage, il suffit d'appeler la classe pour récupérer toutes ces infos. Il faut passer le contexte de l'application ainsi que la "clé" pour récupérer le jwt stocké.

```

var localStorage = LocalStorage(context, datastoreName: "jwt")
val retro = Retro().getRetroClientInstance().create(WatashiApi::class.java)
retro.getUserProductFromFavoris(localStorage.userId, localStorage.jwtToken).enqueue(object :

```

Le système de paiement fonctionne grâce au sdk de Stripe. Stripe permet d'effectuer les achats de manière sécurisé sur différents moyen de paiement (ici googlePay et carte bancaire)

Une clé utilisateur est installé dans le code et une autre clé est installé dans le back de WatashiHouse. Lors d'un achat une requête est envoyé vers le back avec la clé utilisateur ainsi que le montant de la commande, et le back retourne une clé de confirmation qui est envoyé à stripe coté mobile.

Le code pour valider le paiement est disponible dans le fichier ShoppingCart/ValidateShopping.kt