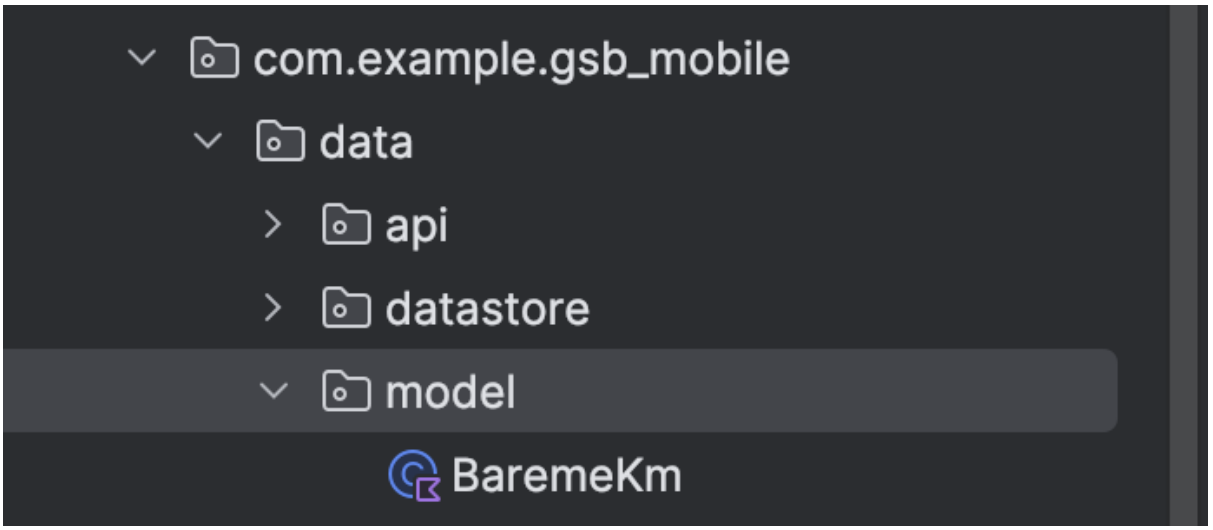
Comment faire un CRUD version Kotlin ?

Pré-requis :

- S'assurer la connexion avec l'application web dans l'API Config
- Avoir créé le DTO dans le projet Symfony ainsi que les invoke response

Etape 1 : Créer la class Kotlin

Dans data/modele créer un fichier "Baremekm"



Ce fichier comportera les même informations que Symfony sous la forme : "
data class {Nom de la classe} (
val {nom du champs} {type}
) "

```kotlin
package com.example.gsb_mobile.data.model

14 Usages
data class BaremeKm (
    val id: Int?,
val bareme: String?,
val puissance_fiscale: Int?,
val annee: Int?,
val km_min:Int?,
val km_max: Int?,
)
```

Etape 2 : ApiService

Il faut désormais configurer le service de récupération des informations en fonction des invoke response.
Plusieurs méthodes sont nécessaires :
getBaremes pour la récupération des barèmes
editBareme pour la modification
newBareme pour la création

GetBareme :

- On récupère le token JWT d'authentification pour vérifier la session
- On construit la requête qui pointe vers Symfony en utilisant la BASE_URL configuré dans ApiConfig ainsi que le token
- On gère le succès de la réponse mais aussi l'erreur (= renvoyer une liste vide pour ne pas faire fuiter de données)
- On parcourt chaque objet JSON dans le tableau pour créer l'objet barème avec tous ses champs.

```
fun getBaremes (context: Context, callback:(List<BaremeKm>?)
→ Unit) {

   CoroutineScope(Dispatchers.IO).launch {

       val token = TokenManager.getToken(context) ?:
return@launch

       val request = Request.Builder()
           .url("${ApiConfig.BASE_URL}/api/bareme")
           .addHeader("Authorization", "Bearer $token")
           .build()

       client.newCall(request).enqueue(object : Callback {

           override fun onFailure(call: Call, e: IOException)
{
               callback(emptyList())
           }

           override fun onResponse(call: Call, response:
Response) {
```

```
                val body = response.body?.string() ?: return
callback(emptyList())
                if (!body.trim().startsWith("[")) {
                    callback(emptyList())
                    return
                }
                val array = JSONArray(body)
                val baremes = mutableListOf<BaremeKm>()

                for (i in 0 until array.length()) {

                    val obj = array.getJSONObject(i)

                    baremes.add(
                        BaremeKm(
                            id = obj.optInt("id"),
                            bareme = obj.optString("bareme"),
                            puissance_fiscale =
obj.optInt("puissance_fiscale"),
                            annee = obj.optInt("annee"),
                            km_min = obj.optInt("km_min"),
                            km_max = obj.optInt("km_max"),

                            )
                    )
                }

                callback(baremes)
            }
        })
    }
}
```

EditBareme :

- Même principe que GetBareme pour la gestion du token
- On crée un objet JSON puis on le convertit avec application/json
- Requête “PUT” plutôt que GET pour la requête
- Gestion de l’echec (= aucune modification si problème existant)
- Gestion du succès (modification OK)

```
fun editBareme(
    context: Context,
    id: Int?,
    bareme: String?,
    puissance_fiscale: Int?,
    annee: Int?,
    km_min: Int?,
    km_max: Int?,
    callback: () -> Unit

) {

    CoroutineScope(Dispatchers.IO).launch {

        val token = TokenManager.getToken(context) ?:
return@launch

        val json = JSONObject()
        json.put("bareme", bareme)
        json.put("puissance_fiscale", puissance_fiscale)
        json.put("annee", annee)
        json.put("km_min", km_min)
        json.put("km_max", km_max)

        val body = json.toString()
            .toRequestBody("application/json".toMediaType())

        val request = Request.Builder()
            .url("${ApiConfig.BASE_URL}/api/bareme/edit/$id")
            .put(body)
            .addHeader("Authorization", "Bearer $token")
            .build()

        client.newCall(request).enqueue(object : Callback {

            override fun onFailure(call: Call, e: IOException)
{}
```

```
            override fun onResponse(call: Call, response:
Response) {
                if (response.isSuccessful) {
                    callback()
                }
            }
        })
    }
}
```

NewBareme :

- Même récupération de token
- Construction du JSON identique à l'edit (application/json)
- Une request en POST pour le new avec l'url qui pointe vers /bareme/new
- Même principe que pour l'edit, aucune modification ni ajout.
- Gestion du succès (mettre à jour l'application avec la nouvelle donnée)

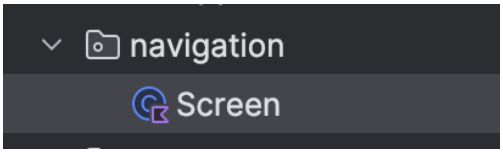
```
fun newBareme(
    context: Context,
    bareme: String,
    puissanceFiscale: Int,
    annee: Int,
    kmMin: Int,
    kmMax: Int,
    callback: () -> Unit
) {
    CoroutineScope(Dispatchers.IO).launch {

        val token = TokenManager.getToken(context) ?:
return@launch

        val json = JSONObject()
        json.put("bareme", bareme)
        json.put("puissance_fiscale", puissanceFiscale)
        json.put("annee", annee)
        json.put("km_min", kmMin)
        json.put("km_max", kmMax)

        val body = json.toString()
            .toRequestBody("application/json".toMediaType())
```

```
        val request = Request.Builder()
            .url("${ApiConfig.BASE_URL}/api/bareme/new")
            .post(body)
            .addHeader("Authorization", "Bearer $token")
            .build()

        client.newCall(request).enqueue(object : Callback {

            override fun onFailure(call: Call, e: IOException)
{
            }

            override fun onResponse(call: Call, response:
Response) {


                if (response.isSuccessful) {
                    CoroutineScope(Dispatchers.Main).launch {
                        callback()
                    }
                }
            }
        })
    }
}
```

Etape 3 : Navigation

Il faut “prévenir” l’application de la création des prochaines pages et leur utilité.

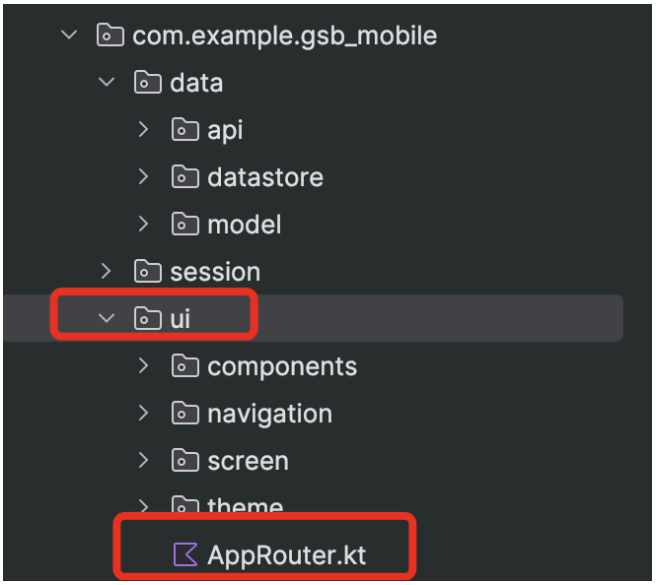
Tout d’abord, si le fichier n’est pas présent, créer le fichier navigation/Screen



Dans ce fichier on indique les différentes pages/routes via des objects pour préparer la navigation entre les pages

```
sealed class Screen {

object Baremes : Screen()
data class BaremeDetail(val bareme: BaremeKm) : Screen()
data class BaremeEdit(val bareme: BaremeKm) : Screen()
object BaremeNew : Screen()
}
```

C'est l'AppRouter qui va s'en servir :
Créer le fichier AppRouter dans le package "ui"



Ce fichier va gérer chaque condition en fonction du fichier Screen :

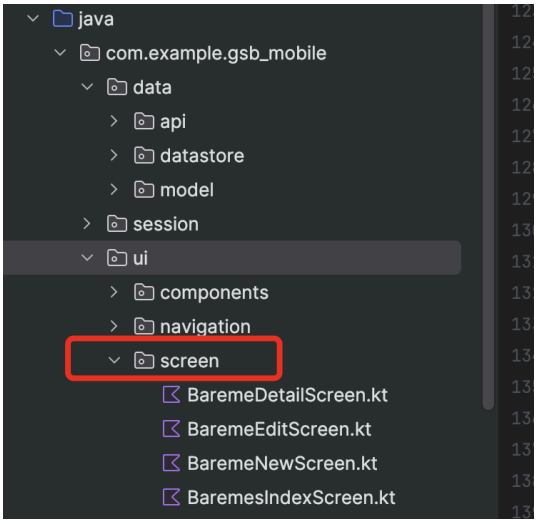
```
package com.example.gsb_mobile.ui

import androidx.compose.runtime.*
import androidx.compose.ui.platform.LocalContext
import kotlinx.coroutines.launch
import com.example.gsb_mobile.data.api.ApiService
import com.example.gsb_mobile.data.api.Session
import com.example.gsb_mobile.data.datastore.TokenManager
import com.example.gsb_mobile.ui.components.AppScaffold
import com.example.gsb_mobile.ui.navigation.Screen
import com.example.gsb_mobile.ui.screen.*


2 Usages
@Composable
fun AppRouter() {

    val context = LocalContext.current
    val scope = rememberCoroutineScope()
```

On va donc préparer chaque page (index, show, edit, new)

```
Screen.Baremes → AppScaffold(
    groupName = groupName,
    onHome = { navigate(Screen.Home) },
    onMentions = { navigate(Screen.Legal) },
    onLogout = { logout() }
) {
    BaremesIndexScreen(
        onVoirDetail = { bareme →
navigate(Screen.BaremeDetail(bareme)) },
        onEditBareme = { bareme →
navigate(Screen.BaremeEdit(bareme)) },
        onNewBareme = { navigate(Screen.BaremeNew) }
    )
}

is Screen.BaremeDetail → AppScaffold(
    groupName = groupName,
    onHome = { navigate(Screen.Home) },
    onMentions = { navigate(Screen.Legal) },
    onLogout = { logout() }
) {
    BaremeDetailScreen(
        bareme = screen.bareme,
        onBack = { navigate(Screen.Baremes) }
    )
}
is Screen.BaremeEdit → AppScaffold(
    groupName = groupName,
    onHome = { navigate(Screen.Home) },
    onMentions = { navigate(Screen.Legal) },
    onLogout = { logout() }
) {
    BaremeEditScreen(
        baremeInitial = screen.bareme,
        onBack = { navigate(Screen.Baremes) }
    )
```

```
}
is Screen.BaremeNew → AppScaffold(
    groupName = groupName,
    onHome = { navigate(Screen.Home) },
    onMentions = { navigate(Screen.Legal) },
    onLogout = { logout() }
) {
    BaremeNewScreen(
        onBack = { navigate(Screen.Baremes) }
    )
}
```

Etape 4 : CRUD
Dans le dossier “screen” créer les fichiers nécessaires au CRUD



BaremeDetailScreen, BaremeEditScreen,BaremeNewScreen,BaremeIndexScreen

BaremeIndexScreen :
L’index crée la fonction puis récupère toutes les entrées dans le tableau de barèmes et prépare chaque bouton présent sur la page (voir le détail, création et modification)

```
package com.example.gsb_mobile.ui.screen

import ...

1 Usage
@Composable
fun BaremesIndexScreen(
    onVoirDetail: (BaremeKm) -> Unit,
    onEditBareme: (BaremeKm) -> Unit,
    onNewBareme: () -> Unit
) {
    val context = LocalContext.current
    var baremes by remember { mutableStateOf<List<BaremeKm>>( value = emptyList()) }
    var loading by remember { mutableStateOf( value = true) }
```

On récupère donc toutes les entrées du tableau :

```
items(baremes) { bareme →
   Card(
       modifier = Modifier
           .fillMaxWidth()
           .padding(vertical = 8.dp),
       elevation = CardDefaults.cardElevation(4.dp)
   ) {
       Column(Modifier.padding(16.dp)) {
           Text(
               text = "Barème : ${bareme.bareme} —
${bareme.puissance_fiscale} CV",
               style = MaterialTheme.typography.titleMedium
           )
           Spacer(Modifier.height(4.dp))
           Text(
               text = "Année : ${bareme.annee ?: "—"}",
               style = MaterialTheme.typography.bodyMedium
           )
           Spacer(Modifier.height(12.dp))
           Button(
               onClick = { onVoirDetail(bareme) },
               modifier = Modifier.fillMaxWidth()
           ) {
               Text("Voir le détail")
           }
```

```
            Spacer(Modifier.height(4.dp))
            Button(
                onClick = { onEditBareme(bareme) },
                modifier = Modifier.fillMaxWidth()
            ) {
                Text("Modifier")
            }
        }
    }
}
```

BaremeDetailScreen :
Dans le même principe, le détail récupère une entrée spécifique du tableau pour afficher le détail du barème

```
@Composable
fun BaremeDetailScreen(
    bareme: BaremeKm,
    onBack: () → Unit
) {

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {

        OutlinedButton(
            onClick = onBack,
            modifier = Modifier.fillMaxWidth()
        ) {
            Text("Retour")
        }

        Spacer(Modifier.height(20.dp))

        Text(
            text = "Détail barème",
            style = MaterialTheme.typography.headlineSmall,
```

```
            fontWeight = FontWeight.Bold
        )

        Spacer(Modifier.height(16.dp))

        Text("Année : ${bareme.annee}")
        Text("Barème : ${bareme.bareme}")
        Text("Puissance Fiscale : ${bareme.puissance_fiscale}")
        Text("Km minimum : ${bareme.km_min}")
        Text("Km max : ${bareme.km_max}")

        }
```

BaremeEditScreen :
L'edit prend en compte les données initiales

```
val context = LocalContext.current
var bareme by remember { mutableStateOf(baremeInitial.bareme
?: "") }
var puissance by remember {
mutableStateOf(baremeInitial.puissance_fiscale?.toString() ?:
"") }
var annee by remember {
mutableStateOf(baremeInitial.annee?.toString() ?: "") }
var km_min by remember {
mutableStateOf(baremeInitial.km_min?.toString() ?: "") }
var km_max by remember {
mutableStateOf(baremeInitial.km_max?.toString() ?: "") }
var errorMessage by remember { mutableStateOf<String?>(null) }
```

puis les affiche

```
OutlinedTextField(
   value = bareme,
   onValueChange = { bareme = it },
   label = { Text("Barème") },
   keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Decimal),
   modifier = Modifier.fillMaxWidth()
)
Spacer(modifier = Modifier.height(12.dp))
```

```
OutlinedTextField(
    value = puissance,
    onValueChange = { puissance = it },
    label = { Text("Puissance Fiscale") },
    keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
    modifier = Modifier.fillMaxWidth()
)
Spacer(modifier = Modifier.height(12.dp))
```

etc. pour chaque donnée dans l'entité

Ensuite, on permet le remplacement de la donnée uniquement si c'est une donnée numérique

```
Button(
    onClick = {
        val puissanceInt = puissance.toIntOrNull()
        val anneeInt = annee.toIntOrNull()
        val kmMinInt = km_min.toIntOrNull()
        val kmMaxInt = km_max.toIntOrNull()

        if (puissanceInt == null || anneeInt == null ||
kmMinInt == null || kmMaxInt == null) {
            errorMessage = "Tous les champs numériques doivent
être des nombres valides."
        } else {
            errorMessage = null
            ApiService.editBareme(
                context,
                baremeInitial.id ?: 0,
                bareme,
                puissanceInt,
                anneeInt,
                kmMinInt,
                kmMaxInt,
            ) { onBack() }
        }
```

```
    },
    modifier = Modifier.fillMaxWidth()
) {
    Text("Enregistrer")
}
```

On enregistre.

BaremeNewScreen :
On crée des données à vide

```
var bareme by remember { mutableStateOf("") }
var puissance by remember { mutableStateOf("") }
var annee by remember { mutableStateOf("") }
var kmMin by remember { mutableStateOf("") }
var kmMax by remember { mutableStateOf("") }
var errorMessage by remember { mutableStateOf<String?>(null) }
```

Ensuite on prépare les champs pour écrire :

```
Column(
    modifier = Modifier
        .fillMaxSize()
        .verticalScroll(rememberScrollState())
        .padding(24.dp)
) {
    Text("Nouveau barème", style =
MaterialTheme.typography.headlineMedium)
    Spacer(modifier = Modifier.height(24.dp))

    OutlinedTextField(
        value = bareme,
        onValueChange = { bareme = it },
        label = { Text("Barème") },
        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Decimal),
        modifier = Modifier.fillMaxWidth()
    )
    Spacer(modifier = Modifier.height(12.dp))

    OutlinedTextField(
        value = puissance,
```

```
        onValueChange = { puissance = it },
        label = { Text("Puissance Fiscale") },
        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
        modifier = Modifier.fillMaxWidth()
    )
    Spacer(modifier = Modifier.height(12.dp))
```

etc. pour chaque donnée dans l'entité

Enfin, on permet l'enregistrement de la nouvelle donnée uniquement si ce sont des données numériques

```
Button(
    onClick = {
        val puissanceInt = puissance.toIntOrNull()
        val anneeInt = annee.toIntOrNull()
        val kmMinInt = kmMin.toIntOrNull()
        val kmMaxInt = kmMax.toIntOrNull()

        if (puissanceInt == null || anneeInt == null ||
kmMinInt == null || kmMaxInt == null) {
            errorMessage = "Tous les champs numériques doivent
être des nombres valides."
        } else {
            errorMessage = null
            ApiService.newBareme(
                context,
                bareme,
                puissanceInt,
                anneeInt,
                kmMinInt,
                kmMaxInt,
            ) { onBack() }
        }
    },
    modifier = Modifier.fillMaxWidth()
) {
    Text("Créer")
```

```
}
```

Etape 5 : Page Home

Enfin on doit restreindre le CRUD aux administrateurs fonctionnels :

Dans le fichier HomeScreen qui gère la page d'accueil on ajoute :

```
if (groupName == "Administrateur Fonctionnel") {
   Spacer(Modifier.height(24.dp))
   NavCard("Baremes", Icons.Default.LocationOn, "Barèmes",
onBaremesClick)
}
```

Cela va permettre de restreindre aux administrateurs et d'ajouter la carte de navigation vers la page de barèmes.