

Laboratoire 10

Navigation avec les fragments Utilisation des services web avec Retrofit

Objectifs d'apprentissage

- Navigation avec les fragments
- Exploitation des services web

Dans ce laboratoire, vous allez apprendre à communiquer avec les services Web en utilisant la librairie **Retrofit**. Vous allez utiliser les **RecyclerView**, **ViewModel** et **LiveData** dans le prolongement de ce que vous aviez appris aux laboratoires précédents. Vous allez utiliser certaines librairies que vous devez configurer dans votre fichier **build.gradle**.

Vous allez utiliser les **coroutines** pour gérer les activités en arrière-plan telles que la communication avec les services web.

Vous allez utiliser la librairie **Gson** de Google pour faire le parsing de vos réponses du format JSON vers le modèle objet.

Vous allez utiliser la librairie **Glide** pour l'affichage de vos images dans les **imageViews**. Cette application se connecte à un service Web pour récupérer et afficher dans un **RecyclerView** les News à partir de l'api <https://newsapi.org>.

Créez un nouveau projet avec **Android Studio** en utilisant le **template Empty Activity**.

Ajoutez le plugin **kotlin-kapt** à votre fichier **build.gradle (Module)**.

Ajoutez le plugin **kotlin-android-extensions** à votre fichier **build.gradle (Module)** s'il n'est pas présent.

Ajoutez les dépendances suivantes à votre fichier **build.gradle** et lancez la synchronisation.

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.squareup.okhttp3:logging-interceptor:3.12.1'
//Glide
implementation 'com.github.bumptech.glide:glide:4.11.0'
kapt 'com.github.bumptech.glide:compiler:4.11.0'

//Coroutines Lifecycle Scopes
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'
implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.2.0'
```

Dans votre fichier **AndroidManifest**, ajoutez la permission internet à votre application puisque vous allez utiliser les services web.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Votre application doit contenir 4 fragments que vous créerez avec les noms suivants **BreakingNewsFragment**, **SavedNewsFragment**, **AllNewsFragment**, **ArticleFragment**.

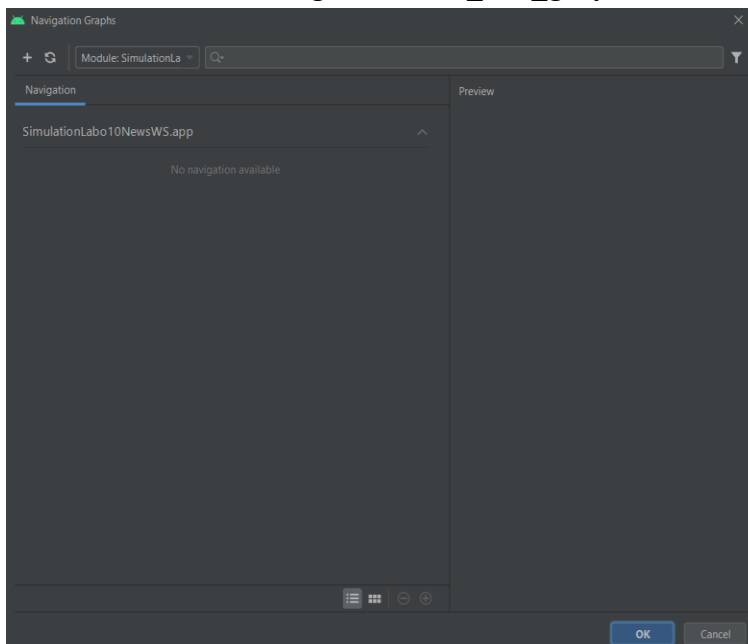
Vous allez vous connecter à Internet pour visualiser des nouvelles à partir de l'api <https://apinews.org>.

Modifiez le fichier **activity_main.xml** pour créer la destination de départ comme suit.

Supprimer le **textview HelloWorld** qui est affiché par défaut.

Ajoutez un widget de type **NavHostFragment** à votre activité.

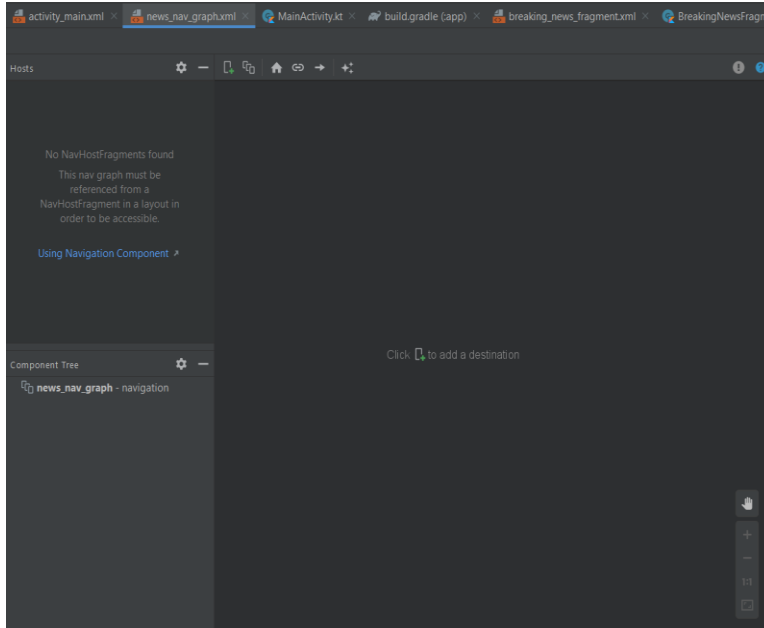
Une fenêtre de graphes de navigation s'affiche. Cliquez sur le bouton **+** et ajoutez un nouveau fichier de navigation **news_nav_graph.xml**.




L'éditeur de navigation du fichier s'affiche.

Cette fenêtre est composée de 3 parties:

- À gauche, la liste des destinations du graphe.
- Au centre, la partie où l'on peut voir et éditer le graphe.
- À droite, les attributs des éléments du graphe.



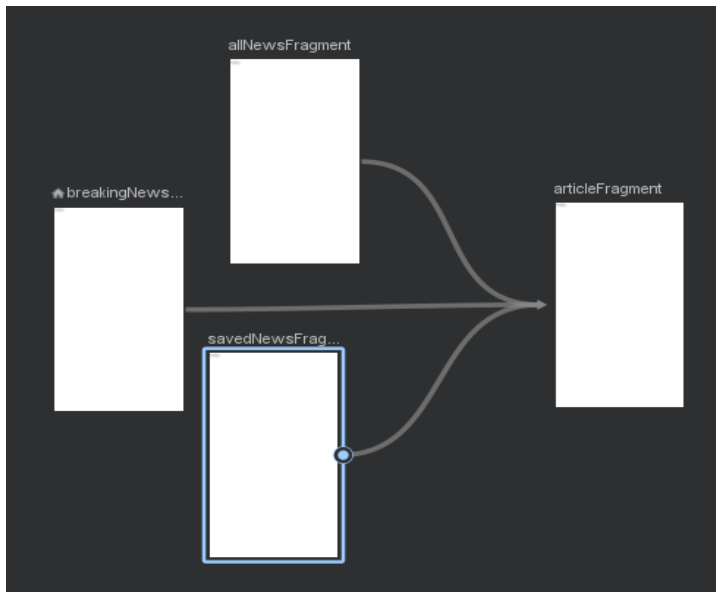
Cliquez sur l'icône  pour ajouter le fragment **BreakingNewsFragment** comme fragment de destination de départ (**startDestination Fragment**) en utilisant le **template Fragment (with ViewModel)**.

Cliquez à nouveau sur l'icône  pour ajouter les autres fragments **AllNewsFragment**, **SavedNewsFragment** et **ArticleFragment**. En utilisant le **template Fragment (with ViewModel)**.

Ajout d'actions

La navigation d'un fragment à un autre s'effectue en utilisant des actions. Une action contient les données nécessaires pour naviguer d'un fragment à un autre, notamment l'identifiant de la destination, les animations de transition s'il y en a, les arguments à passer à la destination s'il y en a.

Faites un glissé-déposé depuis l'ancre à droite du fragment **BreakingNewsFragment** jusqu'au fragment **ArticleFragment** pour les relier en ajoutant une action. L'ID par défaut est **action_breakingNewsFragment_to_articleFragment** pour cette action. Ajoutez deux autres actions partant des fragments **AllNewsFragment** et **SavedNewsFragment** pour aboutir au fragment **ArticleFragment**.



À noter que vous pouvez aussi créer vos fragments directement à partir de l'éditeur de navigation et les insérer dans le graphe de navigation.

Remarquez l'icône de la petite maison au-dessus de **breakingNewsFragment** qui indique que c'est le fragment de destination de départ (**startDestination**: qui est lancé au démarrage de l'activité).

Ajout d'un hôte de navigation à une activité

Pour finaliser la configuration de la navigation, vous devez créer un fragment de navigation hôte, qui sert de conteneur pour le graphe de navigation. Ce conteneur délèguera la navigation du graphe à son contrôleur de navigation appelé **NavController**.

Ouvrez le fichier **activity_main.xml** et glissez dans le **ConstraintLayout** un widget de type **NavHostFragment**. Une fenêtre s'ouvre pour choisir le fichier de navigation. Sélectionnez le fichier de navigation et cliquez sur Ok.

Modifiez l'id du fragment hôte de navigation en **navHostFragmentContainer**. C'est le fragment dans lequel va se faire la navigation.

Ajoutez les contraintes nécessaires pour que le fragment occupe toute la largeur de la fenêtre.

Votre fichier **activity_main.xml** aura un code comme suit:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

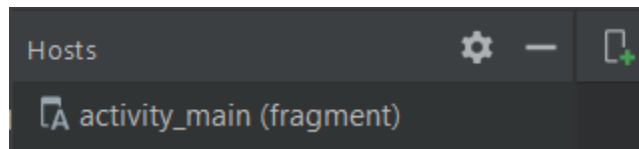
    <fragment
        android:id="@+id/navHostFragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
```

```

        android:layout_width="0dp"
        android:layout_height="0dp"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/news_nav_graph" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Si vous retournez dans l'éditeur de navigation, vous allez voir l'activité dans la section **HOST** du graphe.



Retournez au fichier `activity_main.xml` et ajoutez un widget de type **BottomNavigationView** ayant pour id **bottomNavigationView** au-dessous du fragment **navHostFragment**.

Ajoutez un nouveau fichier de menu **bottom_navigation_menu** avec le contenu suivant. Ce menu de navigation sera affiché en dessous du fragment hôte.

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/savedNewsFragment"
        android:icon="@drawable/ic_favorite"
        android:title="@string/saved_news_txt" />
    <item
        android:id="@+id/searchNewsFragment"
        android:icon="@drawable/ic_all_news"
        android:title="@string/search_news_txt" />
    <item
        android:id="@+id/breakingNewsFragment"
        android:icon="@drawable/ic_breaking_news"
        android:title="@string/breaking_news_txt" />
</menu>

```

Ajoutez les ressources correspondant aux différents fichiers **drawable** (**ic_favorite**, **ic_all_news**, **ic_breaking_news**) à partir de **Vector Asset**.

Ajoutez les chaînes de caractères suivants au fichier **strings.xml**.

```

<string name="breaking_news_txt">Breaking news</string>
<string name="saved_news_txt">Saved news</string>
<string name="search_news_txt">Search news</string>
<string name="article_txt">Article</string>

```

Modifiez l'attribut **menu** du **bottomNavigationView** pour qu'il corresponde au fichier **bottom_navigation_menu**.

Ouvrez le fichier `strings.xml` et ajoutez le contenu qui suit :

```

<string name="hello_breaking_news_fragment">Hello from breaking news
fragment</string>

```

```
<string name="hello_saved_news_fragment">Hello from saved news
fragment</string>
<string name="hello_search_news_fragment">Hello from search news
fragment</string>
<string name="hello_article_fragment">Hello from article fragment</string>
```

Modifiez les labels des différents fragments dans le fichier de navigation avec les chaînes de caractères que vous venez d'insérer dans le fichier strings.xml.

Le fichier res/navigation/news_nav_graph.xml doit ressembler à ce qui suit :

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/news_nav_graph"
    app:startDestination="@id/breakingNewsFragment">

    <fragment
        android:id="@+id/breakingNewsFragment"
        android:name="ca.qc.cgodin.ggnewsproject.BreakingNewsFragment"
        android:label="fragment_breaking_news"
        tools:layout="@layout/fragment_breaking_news" >
        <action
            android:id="@+id/action_breakingNewsFragment_to_articleFragment"
            app:destination="@id/articleFragment" />
    </fragment>
    <fragment
        android:id="@+id/articleFragment"
        android:name="ca.qc.cgodin.ggnewsproject.ArticleFragment"
        android:label="fragment_article"
        tools:layout="@layout/fragment_article" />
    <fragment
        android:id="@+id/savedNewsFragment"
        android:name="ca.qc.cgodin.ggnewsproject.SavedNewsFragment"
        android:label="fragment_saved_news"
        tools:layout="@layout/fragment_saved_news" >
        <action
            android:id="@+id/action_savedNewsFragment_to_articleFragment"
            app:destination="@id/articleFragment" />
    </fragment>
    <fragment
        android:id="@+id/searchNewsFragment"
        android:name="ca.qc.cgodin.ggnewsproject.SearchNewsFragment"
        android:label="fragment_search_news"
        tools:layout="@layout/fragment_search_news" >
        <action
            android:id="@+id/action_searchNewsFragment_to_articleFragment"
            app:destination="@id/articleFragment" />
    </fragment>
</navigation>
```

Explorez le fichier en xml et en mode Design pour bien comprendre comment ça fonctionne.

Identifiez les transitions qui ont été ajoutées qui permettent de passer d'un fragment à un autre.

Lancez votre application et cliquez sur les boutons de navigation du bas.

Remarquez bien que les boutons ne font rien pour l'instant.

Ajoutez la ligne suivante à la fin de la fonction **onCreate** de la classe **MainActivity**.

```
bottomNavigationView.setupWithNavController(navHostFragment.findNavController())
```

Insérez si nécessaire le plugin pour les extensions **kotlin-android-extensions** à votre fichier **build.gradle (app: Module)** pour pouvoir accéder directement aux éléments de vos fichiers sans utiliser la fonction **findViewById** dans vos activités.

Relancez votre application et cliquez à nouveau sur les boutons de navigation du bas.

Ça doit fonctionner maintenant.

Communication avec les services Web

Ajoutez le plugin vous permettant de générer les classes dont vous aurez besoin à partir d'un fichier JSON.

Cliquez sur **File --> Settings --> Plugins**

Saisissez le mot-clé **JsonToKotlinClass** dans la zone de recherche pour chercher dans le MarketPlace. Cliquez sur Install pour faire l'installation.

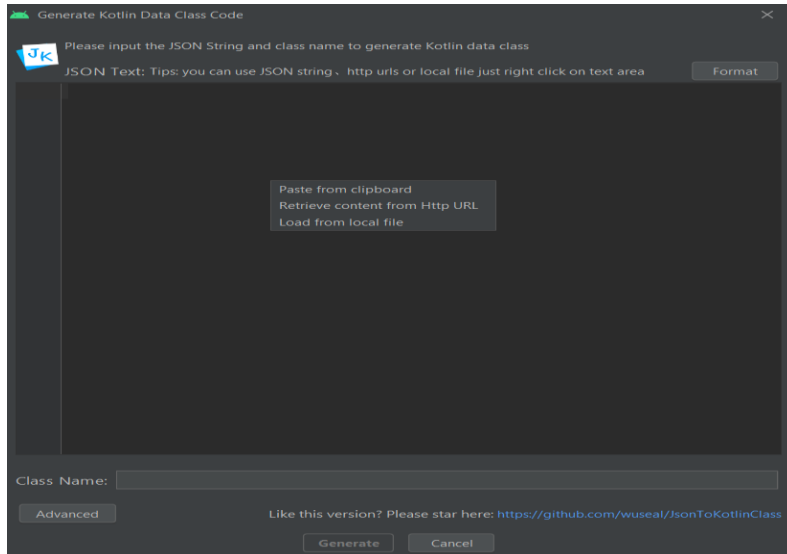
Dans un navigateur, ouvrez le site web <https://newsapi.org> et créez une clé d'api pour pouvoir accéder aux ressources exposées à travers cette api.

Créez les classes dont vous aurez besoin dans un sous package **models**. Une fois le plugin installé, vous pouvez créer les classes en utilisant l'option **File->New-> Kotlin data class File from JSON**.

Faites un clic droit à l'intérieur de la zone de texte et choisissez l'option **Retrieve Content from http URL**.

Saisissez l'url suivant pour générer les classes à partir du contenu JSON.

https://newsapi.org/v2/top-headlines?country=ca&apiKey=votre_apiKey



Ouvrez le package **models** et observez le contenu des classes qui ont été générées. Vous devez en trouver 3 classes: NewsResponse, Source et Article. Vous allez les utiliser plus tard.

Ouvrez le fichier BreakingNewsFragment.kt et créez une variable **viewModel** au début de la classe comme suit:

```
private lateinit var viewModel: NewsViewModel
```

Modifiez le fichier fragment_breaking_news.xml en insérant un recyclerView avec pour id **rvBreakingNews** et l'attribut **app:layoutManager** ayant pour valeur **androidx.recyclerview.widget.LinearLayoutManager** comme suit:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/constraintLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#8BC34A">
```



```

tools:context=".BreakingNewsFragment">

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rvBreakingNews"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Ajoutez le fichier **news_item.xml** avec le contenu suivant dans votre répertoire de layout:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/constraintLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/purple_500">

    <ImageView
        android:id="@+id/ivCoverImage"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:contentDescription="@string/cover_image_txt"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/ivCoverImage"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Créez la classe **NewsAdapter** comme suit pour afficher la liste des articles que vous allez récupérer du service web <https://newsapi.org>.

```
class NewsAdapter() : RecyclerView.Adapter<NewsAdapter.ArticleViewHolder>() {
```

Ajoutez un objet **articles** à la classe **NewsAdapter** pour manipuler la liste de vos articles:

```
private var articles: List<Article> = emptyList()
```

Créez la classe interne **ArticleViewHolder** comme suit dans la classe **NewsAdapter**:

```

inner class ArticleViewHolder(itemView: View) :
    RecyclerView.ViewHolder(itemView) {
    val ivCoverImage: ImageView = itemView.findViewById(R.id.ivCoverImage)
    val tvTitle: TextView = itemView.findViewById(R.id.tvTitle)
}

```

```

        override fun toString(): String {
            return super.toString() + " '" + tvTitle.text + "'"
        }
    }
}

```

Modifiez la fonction **onCreateViewHolder** comme suit:

```

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ArticleViewHolder {
    val view = LayoutInflater.from(parent.context)
        .inflate(R.layout.news_item, parent, false)
    return ArticleViewHolder(view)
}

```

Modifier la fonction **onBindViewHolder** comme suit. Vous utilisez la librairie **Glide** pour afficher les images dans votre **imageView**:

```

override fun onBindViewHolder(holderArticle: ArticleViewHolder, position: Int)
{
    val article = articles[position]
    holderArticle.tvTitle.text = article.title
    holderArticle.itemView.apply {

        Glide.with(this).load(article.urlToImage).into(holderArticle.ivCoverImage)
        tvTitle.text = article.title
    }
}

```

Modifiez la fonction **getItemCount** comme suit:

```

override fun getItemCount(): Int = articles.size

```

Ajoutez la fonction **setArticles** pour affecter une liste d'articles à l'objet **articles** et notifier le **recyclerview** par rapport à la modification.

```

fun setArticles(articles: List<Article>) {
    this.articles = articles
    notifyDataSetChanged()
}

```

Création de l'interface NewsApiService

Pour communiquer avec les services Web, vous allez utiliser la librairie **Retrofit**.

Créez un sous-package **network** et créez une interface **NewsApiService** qui regroupe les fonctions de communication avec le service Web. Notre premier objectif est d'aller chercher les articles qui nous seront retournés sous forme de chaîne de caractères dans le format JSON. Vous allez ajouter à cette classe la méthode **getBreakingNews** comme suit pour ce faire.

```

@GET("v2/top-headlines")
suspend fun getBreakingNews(
    @Query("country")
    countryCode: String = "ca",
    @Query("page")
    pageNumber: Int = 1,
    @Query("apiKey")
    apiKey: String = API_KEY
): Response<NewsResponse>

```

Pour dire à **Retrofit** ce que cette méthode **getBreakingNews** est supposée faire, on utilise l'annotation **@GET** et on spécifie le chemin (**endpoint**) amenant au service web. Dans ce cas précis, le chemin est **v2/top-headlines**. À l'appel de la méthode **getBreakingNews**, Retrofit ajoute le chemin à l'url de base (que vous aurez à définir plus tard) dans le **builder de Retrofit** et retourne un objet de type **NewsResponse** en utilisant le convertisseur qui sera aussi ajouté au **builder de Retrofit**. Ici on utilise une coroutine pour ne pas bloquer l'interface utilisateur.

L'adresse du service web complet deviendra <https://newsapi.org/v2/top-headlines>.

Création de la classe RetrofitInstance

Dans le sous-package **network**, créez une classe **RetrofitInstance** comme suit qui permet d'exposer un objet **retrofitService** pour communiquer avec le service web.

```
class RetrofitInstance {
    companion object {
        private val retrofitInstance by lazy {
            Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .build()
        }

        val retrofitService by lazy {
            retrofitInstance.create(NewsApiService::class.java)
        }
    }
}
```

Cette classe crée en premier lieu un objet Retrofit **retrofitInstance** en utilisant le Builder de la classe Retrofit. Retrofit a besoin d'au moins deux informations pour construire une API de service Web: l'URI de base qu'on lui passe en utilisant la fonction **baseUrl** et un convertisseur des données reçues du format json en objet Kotlin. On lui passe le convertisseur en utilisant la fonction **addConverterFactory**. On fait ensuite appel à la fonction **build** pour générer l'objet **retrofitInstance**.

On utilise ensuite la méthode **create** de l'objet **retrofitInstance** pour créer **by lazy** l'objet **retrofitService** avec l'interface **NewsApiService**. Cet appel étant assez coûteux, vous exposez cet objet au reste de l'application à travers la classe **RetrofitInstance**.

Créez la variable **BASE_URL** au début du fichier avant la déclaration de la classe:

```
private const val BASE_URL = "https://newsapi.org"
```

Appel au service web dans le repository

Ajoutez un sous-package **repository** et créez une classe **NewsRepository** dans ce package. Modifiez la classe **NewsRepository** comme suit:

```
class NewsRepository() {
    suspend fun getBreakingNews(countryCode: String, pageNumber: Int) =
        RetrofitInstance.retrofitService.getBreakingNews(countryCode,
            pageNumber)
}
```

Appel au service web dans le ViewModel en passant par le repository

Modifiez la classe **BreakingNewsViewModel** héritant de la classe **ViewModel** et recevant dans son constructeur une variable de type **NewsRepository**. Cette variable permet d'accéder au service web pour obtenir les breaking news.

```
val breakingNews: MutableLiveData<NewsResponse> = MutableLiveData()
```

La fonction **getBreakingNews**, donnée ci-dessous, et que vous devez ajouter à la classe **BreakingNewsViewModel**, est appelée à la création du **ViewModel**.

```
init {  
    getBreakingNews("ca")  
}
```

Créez la fonction **getBreakingNews** comme suit dans le **ViewModel**:

```
private fun getBreakingNews(countryCode: String) = viewModelScope.launch {  
    try {  
        val response = newsRepository.getBreakingNews(countryCode)  
        breakingNews.postValue(response.body())  
    } catch (e: Exception){  
    }  
}
```

Modification du fragment BreakingNewsFragment

Modifiez le fragment **BreakingNewsFragment** comme suit:

Ajoutez le code suivant a la fin de la fonction **onViewCreated**:

```
val newsRepository = NewsRepository()  
val viewModelProviderFactory = NewsViewModelProviderFactory(newsRepository)  
  
viewModel = ViewModelProvider(this,  
viewModelProviderFactory).get(NewsViewModel::class.java)  
  
rvBreakingNews.adapter = NewsAdapter()  
  
viewModel.breakingNews.observe(  
    viewLifecycleOwner,  
    Observer { newsResponse ->  
        newsAdapter.setArticles(newsResponse.articles)  
    }  
)
```

Vous devez ajouter la classe **NewsViewModelProviderFactory** vous permettant de créer vos objets **NewsViewModel** comme on avait fait au laboratoire précédent. Cette classe expose une fonction **create** vous permettant de créer vos objets. Voici le contenu de cette classe.

```
class NewsViewModelProviderFactory(  
    private val newsRepository: NewsRepository  
) : ViewModelProvider.Factory {  
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {  
        return if(modelClass.isAssignableFrom(NewsViewModel::class.java)){  
            NewsViewModel(newsRepository) as T  
        } else {  
            throw IllegalArgumentException("Unknown ViewModel class")  
        }  
    }  
}
```

Lancez l'application.

Vous devez voir la liste des **BreakingNews** affichée dans le fragment **BreakingNewsFragment**.

Modifiez le programme pour afficher toutes les nouvelles (en utilisant l'url

https://newsapi.org/v2/everything?q=mot_clé&apiKey=votre_apiKey) dans le fragment qui lui est réservé **AllNewsFragment**.

Ajoutez une nouvelle fonction **getSearchNews** comme suit à l'interface **NewsApiService**. Cette fonction vous permet de chercher les Nouvelles par mot-clé que vous passez à la fonction en utilisant le paramètre **searchQuery**.

```
@GET("v2/everything")
suspend fun getSearchNews(
    @Query("q")
    searchQuery: String,
    @Query("apiKey")
    apiKey: String = API_KEY
): Response<NewsResponse>
```

Ajoutez une fonction **getSearchNews** aussi à votre classe **NewsRepository** pour faire appel à la fonction **getSearchNews** de l'interface **NewsApiService** comme vous aviez fait pour la fonction **getBreakingNews**.

```
suspend fun getSearchNews(searchQuery: String) =
    RetrofitInstance.retrofitService.getSearchNews(searchQuery)
```

Utilisation d'un seul ViewModel

Étant donné que la durée de vie d'un **ViewModel** est attachée à la durée de vie de l'activité hébergeante, on peut supprimer les autres **ViewModel** et s'en servir d'un seul qui sera attaché à l'activité. Faites un **refactor** pour renommer le **BreakingNewsViewModel** en **NewsViewModel**.

Dans la classe **MainActivity**, créez une variable **newsViewModel** comme suit:

```
private lateinit var newsViewModel: NewsViewModel
```

Créez une variable **navController** au début de la classe **MainActivity** comme suit:

```
private val navController by lazy {
    findNavController(R.id.navHostFragment)
}
```

À la fin de la fonction **onCreate** de la classe **MainActivity**, ajoutez le code suivant pour initialiser la variable **newsViewModel**.

```
val newsRepository = NewsRepository()
val viewModelProviderFactory = NewsViewModelProviderFactory(newsRepository)

try {
    val viewModelProvider = ViewModelProvider(
        navController.getViewModelStoreOwner(R.id.news_nav_graph),
        viewModelProviderFactory)
    newsViewModel = viewModelProvider.get(NewsViewModel::class.java)
} catch (e: IllegalArgumentException) {
    e.printStackTrace()
}
```

Ici on obtient un **ViewModelProvider** à partir du graphe de navigation (**news_nav_graph**), ce qui entraîne que notre **newsViewModel** sera disponible à travers tous les fragments faisant partie de ce graphe.

Dans votre fragment, supprimez la variable **breakingNewsViewModel** et créez cette nouvelle variable **newsViewModel** au début de la classe **BreakingNewsFragment** pour accéder au ViewModel.

```
private val newsViewModel: NewsViewModel by
navGraphViewModels(R.id.news_nav_graph)
```

Supprimez les lignes suivantes menant à la création du **breakingNewsViewModel** dans la classe **BreakingNewsFragment**.

```
val newsRepository = NewsRepository()
val viewModelProviderFactory = NewsViewModelProviderFactory(newsRepository)

breakingNewsViewModel = ViewModelProvider(this,
viewModelProviderFactory).get(BreakingNewsViewModel::class.java)
```

Allez dans les autres classes de Fragment et créez la variable **newsViewModel** au début de la classe comme vous avez fait pour la classe **BreakingNewsFragment**.

```
private val newsViewModel: NewsViewModel by
navGraphViewModels(R.id.news_nav_graph)
```

Passage de données d'un fragment à un autre (Safe Args)

Pour pouvoir passer un article d'un autre fragment au fragment Article, on va sérialiser la classe Article.

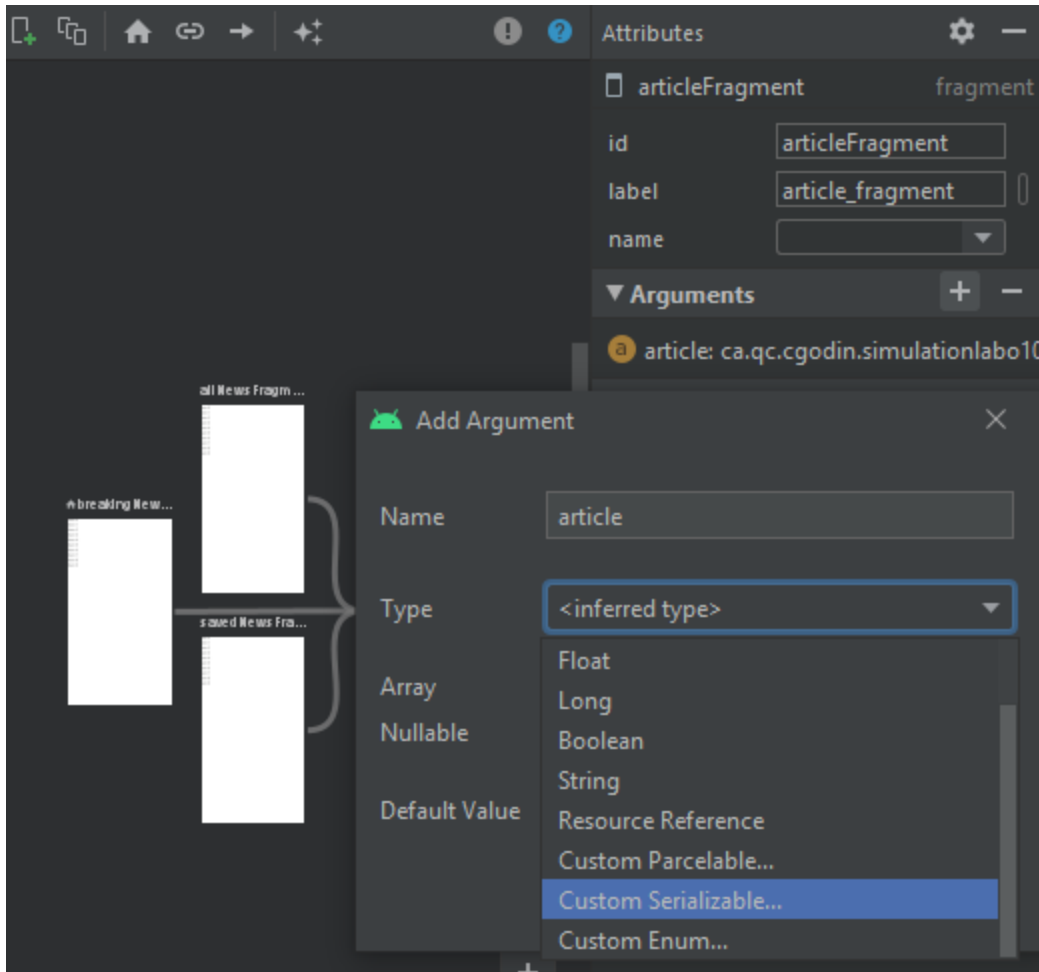
Allez dans la classe Article et modifiez-la comme suit en la faisant implémenter l'interface **Serializable**. Ceci est très important pour pouvoir passer un article au fragment **ArticleFragment**.

```
data class Article(
    @SerializedName("author")
    val author: String,
    @SerializedName("content")
    val content: String,
    @SerializedName("description")
    val description: String,
    @SerializedName("publishedAt")
    val publishedAt: String,
    @SerializedName("source")
    val source: Source,
    @SerializedName("title")
    val title: String,
    @SerializedName("url")
    val url: String,
    @SerializedName("urlToImage")
    val urlToImage: String
) : Serializable
```

Ouvrez votre fichier de graphe de navigation en mode Design.

Sélectionnez le fragment **ArticleFragment**.

Dans la partie de droite, cliquez sur le bouton **+** pour ajouter un argument.



Saisissez **article** dans le champ Name. Et choisissez **Custom Serializable...** dans le champ type. Ensuite cliquer sur **Add**.

Faites un **rebuild** de votre projet.

Modifiez la classe **ArticleFragment** comme suit pour pouvoir afficher un article dans un **WebView**.

Au début de la classe, ajoutez la variable suivante:

```
private val args: ArticleFragmentArgs by navArgs()
```

Ajoutez le code suivant à la fin de la fonction **onViewCreated**:

```
val article = args.article
webView.apply {
    webViewClient = WebViewClient()
    loadUrl(article.url)
}
```

Modification de votre adaptateur

Ajoutez la fonction suivante à votre classe adaptateur pour gérer les clics sur les items.

```
fun setOnItemClickListener(listener: (Article) -> Unit){
    onItemClick = listener
}
```

Créez la variable suivante au début de votre classe **NewsAdapter**

```
private lateinit var onItemClickListener: ((Article) -> Unit)
```

Ajoutez les lignes suivantes à la fin de la fonction **onActivityCreated** de votre classe **BreakingNewsFragment**.

```
newsAdapter.setOnItemClickListener {  
    Log.i("Labo10", "You have clicked on me!!")  
    val bundle = Bundle().apply {  
        putSerializable("article", it)  
    }  
    findNavController().navigate(  
        R.id.action_breakingNewsFragment_to_articleFragment,  
        bundle  
    )  
}
```

Ajoutez les lignes suivantes à la fonction **onBindViewHolder**

```
holderArticle.itemView.setOnClickListener {  
    onItemClickListener(article)  
}
```

Modifiez votre fichier **article_fragment.xml** comme suit:

```
<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:id="@+id/articleConstrainLayout"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".fragments.ArticleFragment">  
    <WebView  
        android:id="@+id/webView"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent" />  
  
    <com.google.android.material.floatingactionbutton.FloatingActionButton  
        android:id="@+id/fab"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:src="@drawable/ic_favorite"  
        app:fabSize="mini"  
        android:layout_marginBottom="8dp"  
        android:layout_marginEnd="12dp"  
        app:layout_constraintBottom_toBottomOf="parent"  
        app:layout_constraintEnd_toEndOf="parent" />  
</androidx.constraintlayout.widget.ConstraintLayout>
```

Lancez votre application pour tester que vous pouvez afficher les articles une fois que vous cliquez dessus.

Travail à faire

Dans votre fragment **AllNewsFragment**, ajoutez un **editText** pour permettre à l'utilisateur de chercher des articles liés à ce mot-clé.

Vous devez afficher ces données dans un **RecyclerView** en dessous de votre **editText**.

Modifiez le programme pour pouvoir sauvegarder dans une base de données avec Room vos nouvelles favorites.

Modifiez le programme pour pouvoir afficher dans le fragment **SavedNewsFragment** vos nouvelles favorites sauvegardées dans la base de données Room.

A remettre dans un fichier zippé au plus tard le jeudi 30 octobre à minuit.

Bon travail !