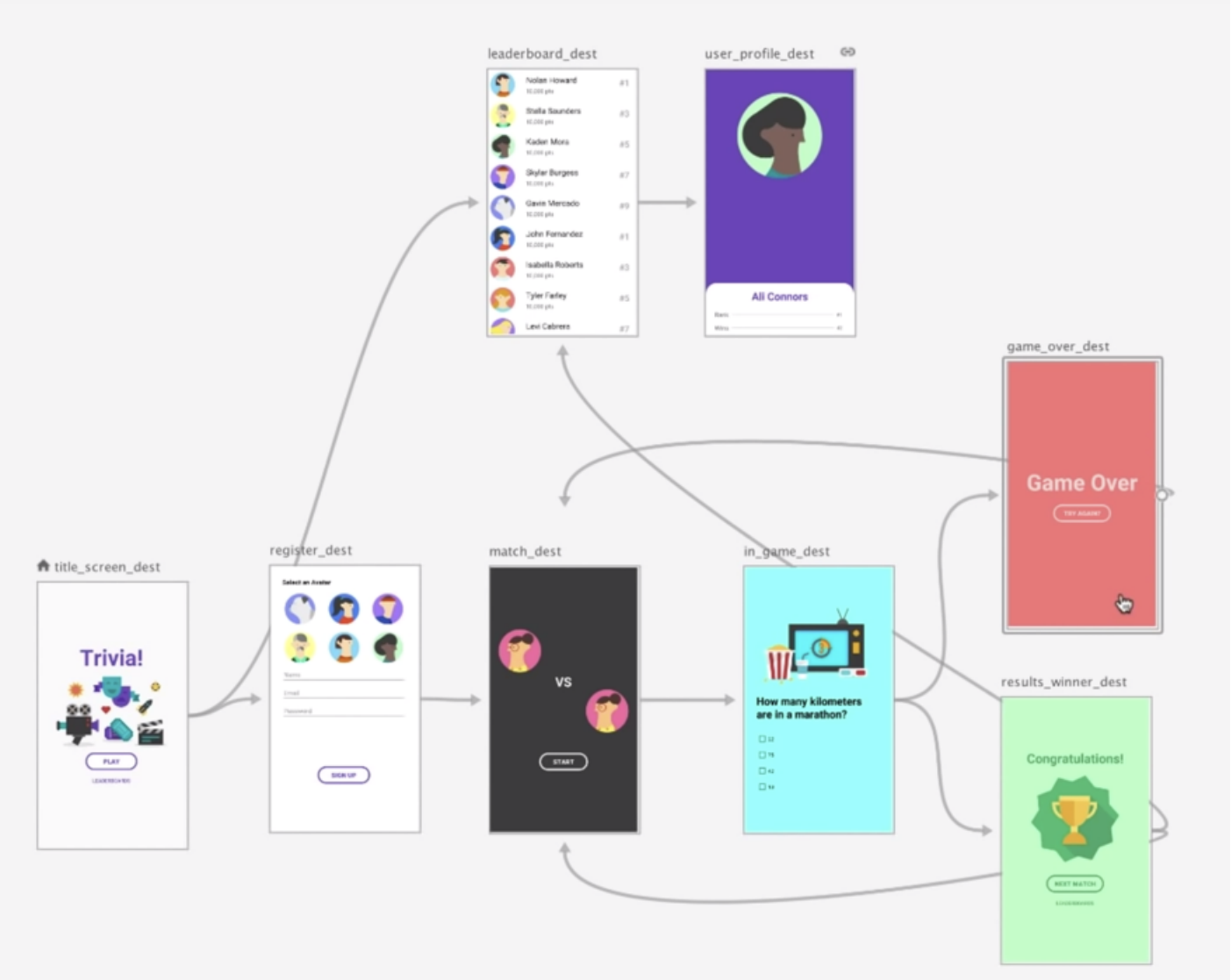
Navigation

La bibliothèque JetPack Navigation va nous permettre de gérer le chemin de l'utilisateur à travers les différents écrans de l'application. La bibliothèque nous permet d'abstraire toute la complexité liée à ce concept en utilisant des mécaniques de génération de code. Cela va donc simplifier beaucoup pour le développeur :

- La gestion de la pile des écrans
- La nagivation arrière
- Le passage des arguments
- Les animations de transition

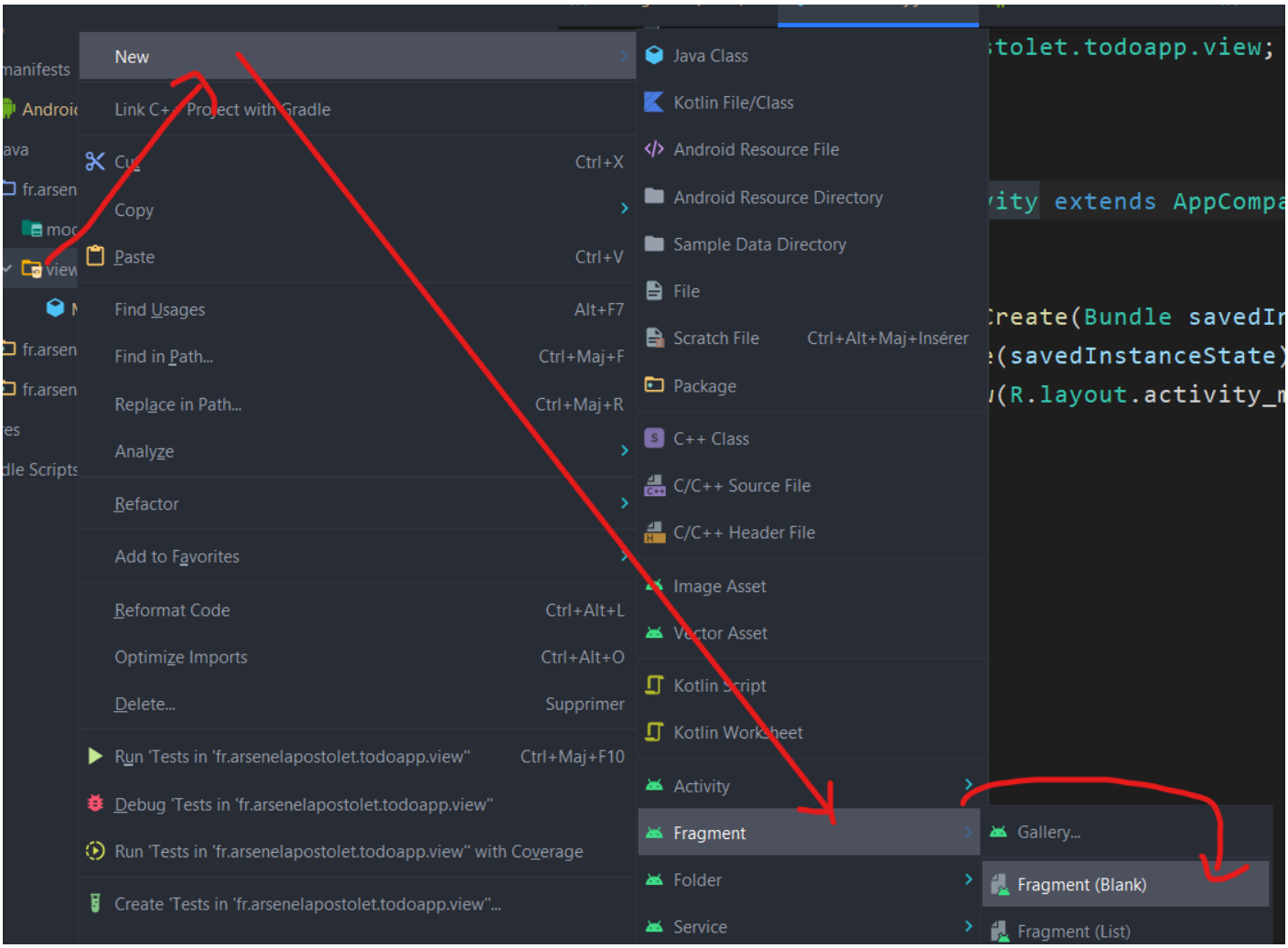
On va donc pouvoir tisser les liens entre les écrans utilisée une interface d'Android studio qui ressemble un peu à ce qu'on pourrait retrouver sur des applications de conception graphique appelé le Navigation Graph :



Utilisation de la naviation

Créer les Fragments

Pour commencer nous allons créer nos différents écrans sous la forme de Fragments. Un fragment est un compostant d'interface graphique que nous allons utiliser à la manière d'une page de notre application. Nous allons avoir besoin de deux écrans : un pour afficher notre liste de Todos, et un pour afficher les détails d'un Todo. Dans le package `view` faites `Clic droit > New > Fragment > Blank Fragment` :



Cela vous nous créer deux fichiers :

- `TodoListFramgent.java` : le fichier "code behind" de notre Fragment, qui va permettre de décrire la logique propre à l'interface graphique
- `fragment_todo_list.xml` : le fichier de Layout, qui va décrire déclarativement, comme un template, notre interface graphique.

Dans `TodoListFramgent` vous pouvez supprimer toutes les méthodes et commentaires à l'exception de `onCreateView`. Ils ne seront pas utiles pour l'instant et on y voit plus clair :

```
public class TodoListFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
    }
}
```

La méthode `onCreateView` est appelée par le framework lorsqu'il va initialiser notre Fragment. Elle contient pour l'instant que la ligne suivante :

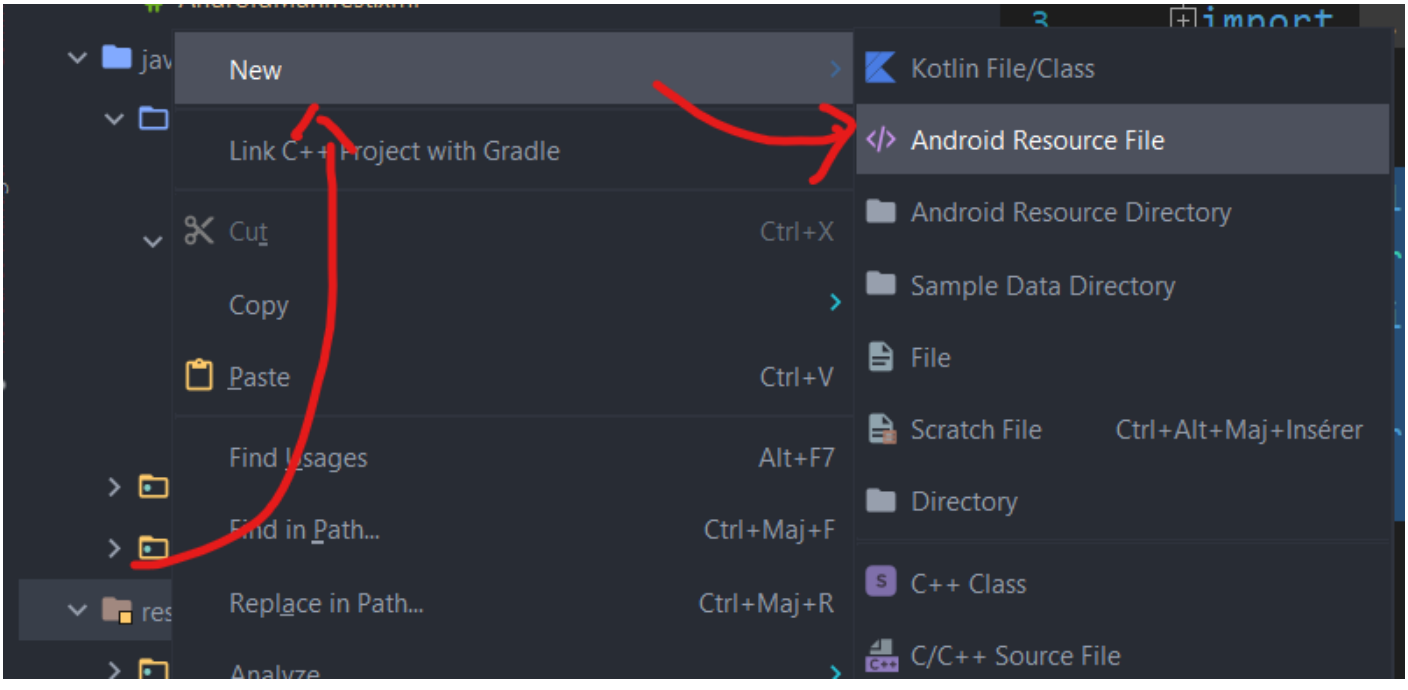
```
return inflater.inflate(R.layout.fragment_todo_list, container, false);
```

Cette ligne va *inflate* (= gonfler en anglais) notre fichier de Layout XML afin de le transformer en objet Java contenant notre vue.

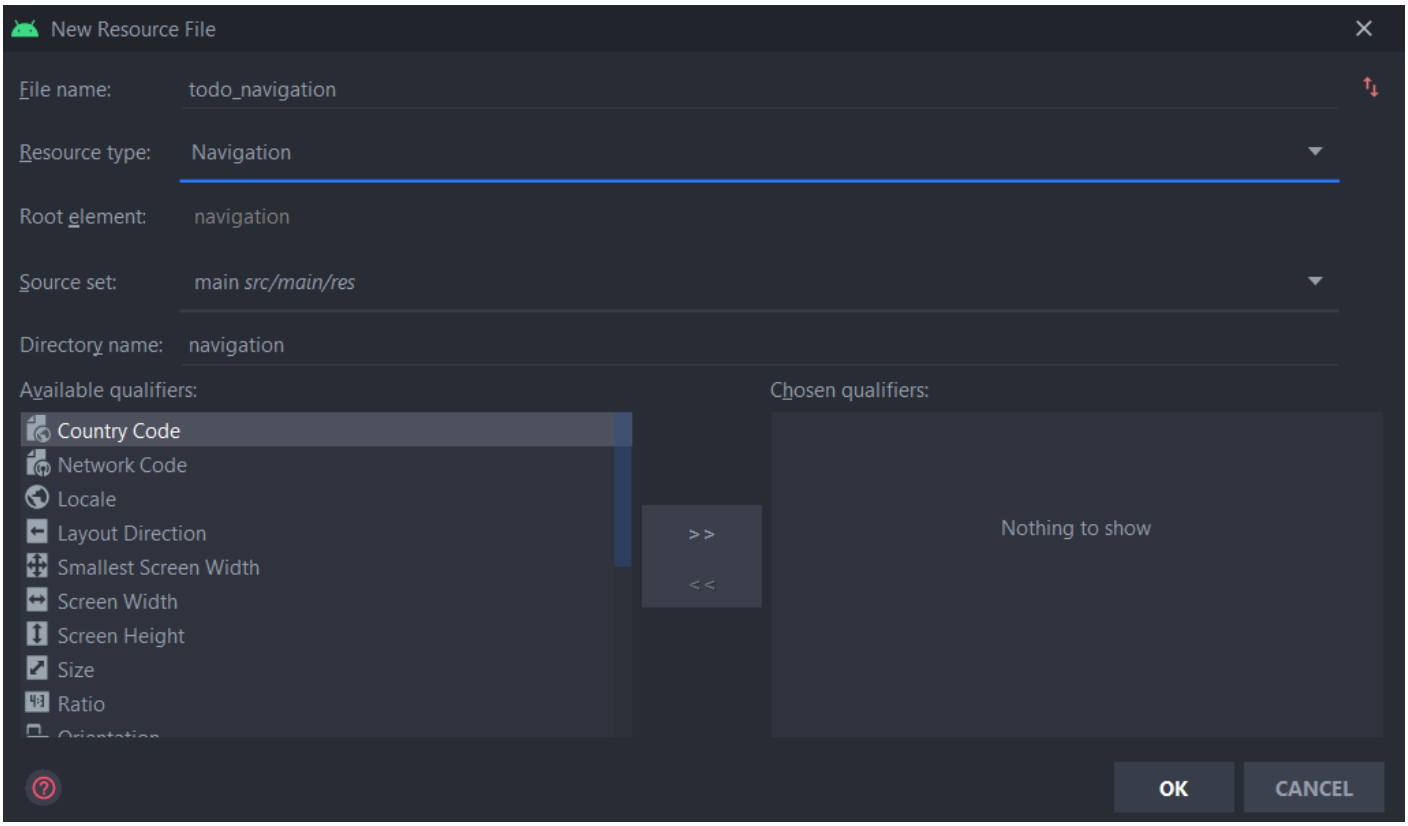
Répétez l'opération pour créer un fragment pour le détail d'un Todo. Vous pouvez également modifier la valeur des `TextView` dans chacun des Layouts de nos Fragment par "Hello List Fragment" et "Hello Detail Fragement" afin de pouvoir les différencier quand on testera la Navigation.

Créer la navigation

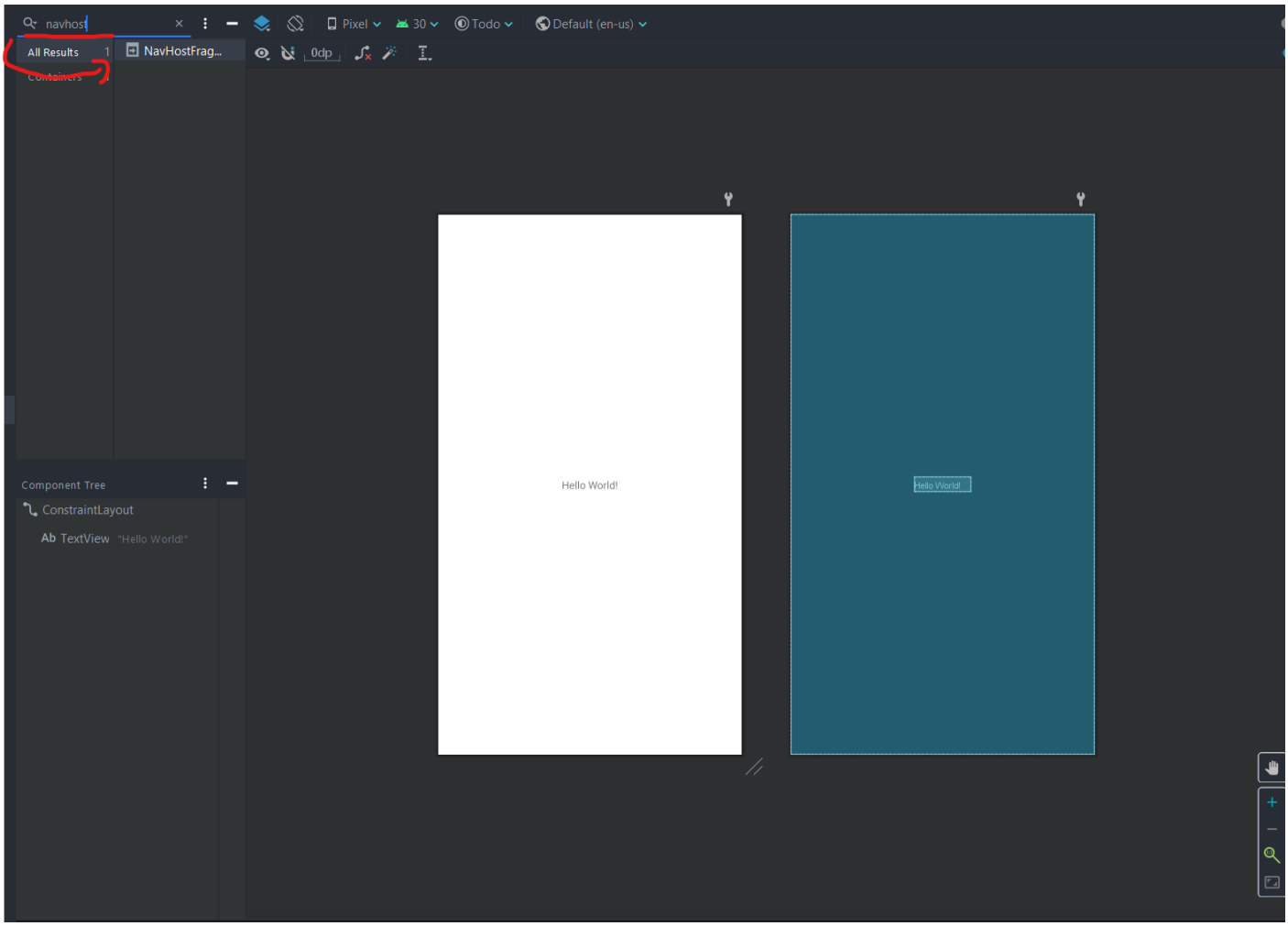
On va maintenant créer la navigation. Sur le dossier `res` faites `Clic doit > New > Android Resource File` :



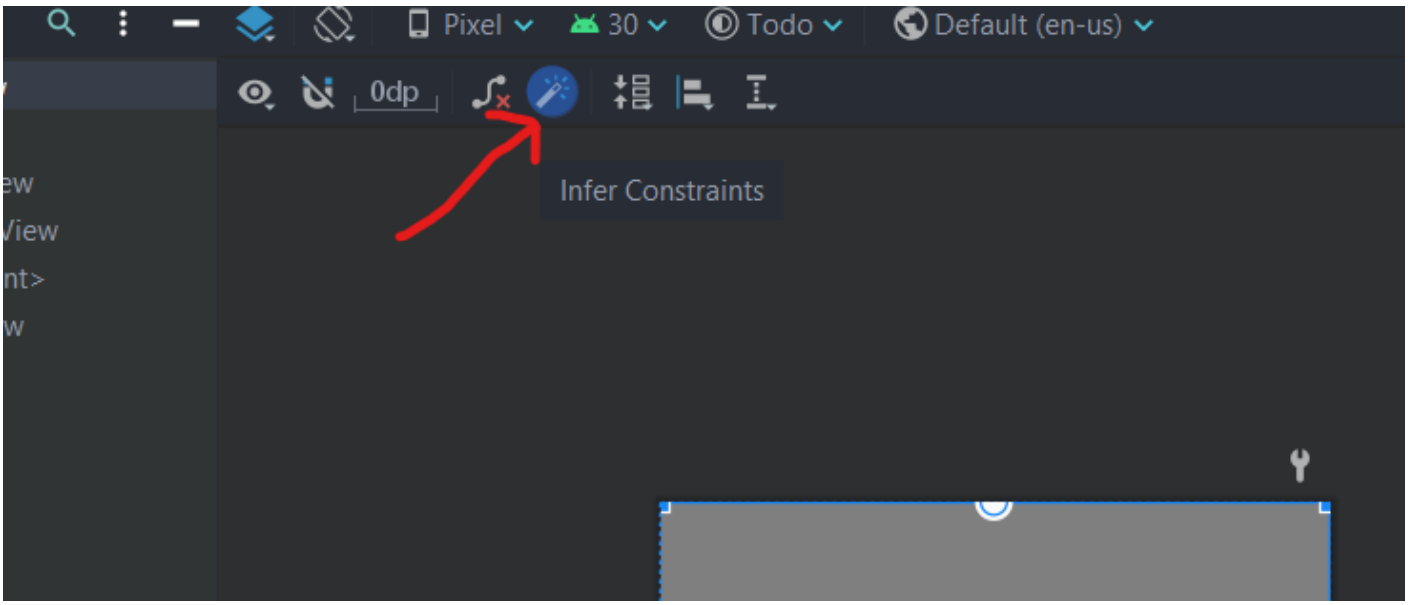
N'oubliez pas de séléctionner le Type de ressource "Navigation" :



On a donc maintenant notre interface de conception de Navigation. Cependant elle n'est pas encore connectée à notre application. Nous allons devoir configurer notre `MainActivity` comme hôte de notre navigation. Pour ce faire, ouvrez le fichier de Layout de notre `MainActivity` situé à `res/layout/activity_main.xml`. Dans l'interface de conception, cherchez un controle appelé `NavHostFragment` :

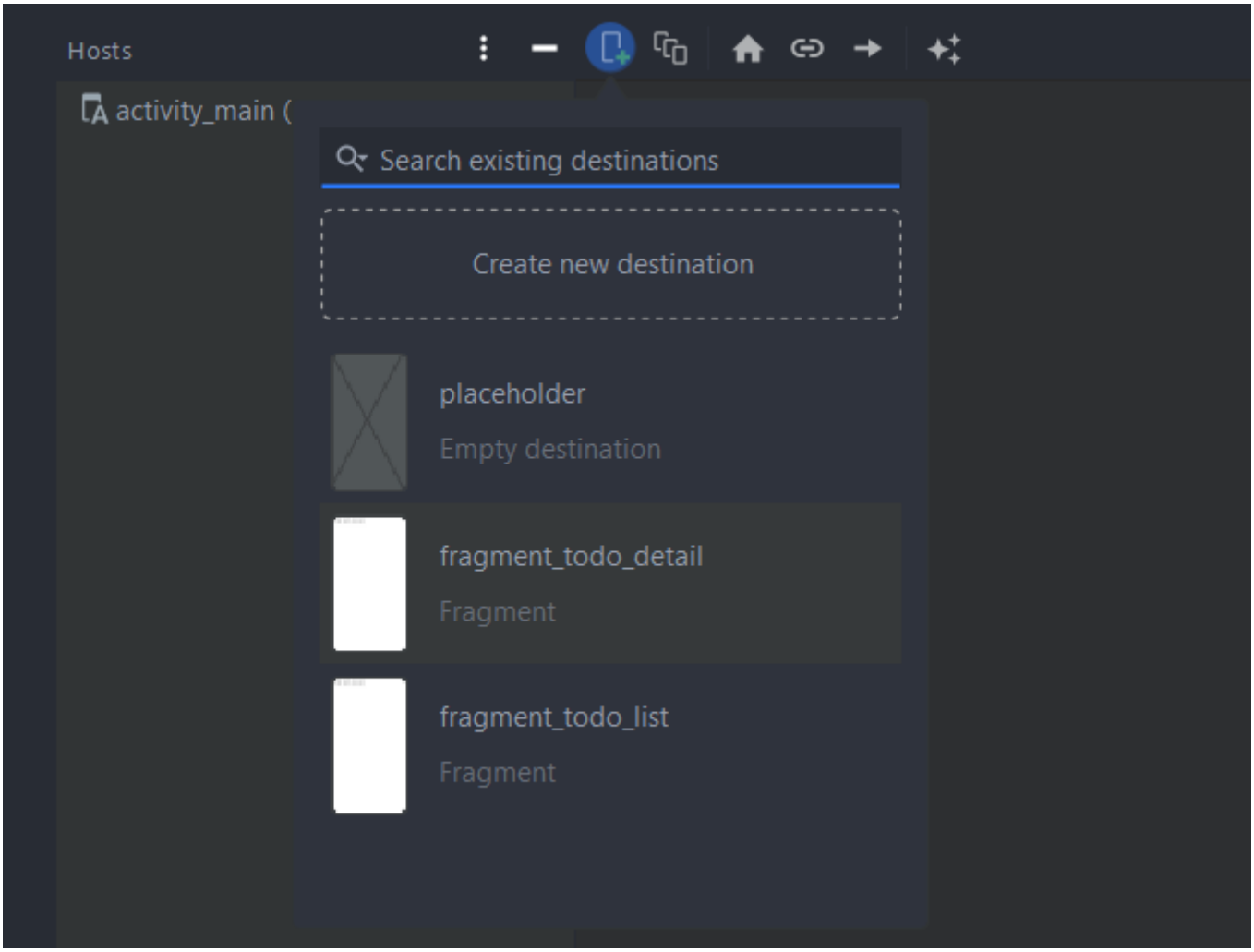


Remplacez le texte "Hello World!" par ce controle. Un graphe de navigation vous est demandé, choisissez celui que nous venont de créer : `todo_navigation`. Ensuite, utilisez le bouton "Infer Constraint" pour configurer autmatiquement les contraintes du Constraints Layout, sur lequel nous reviendrons plus en détail plus tard :

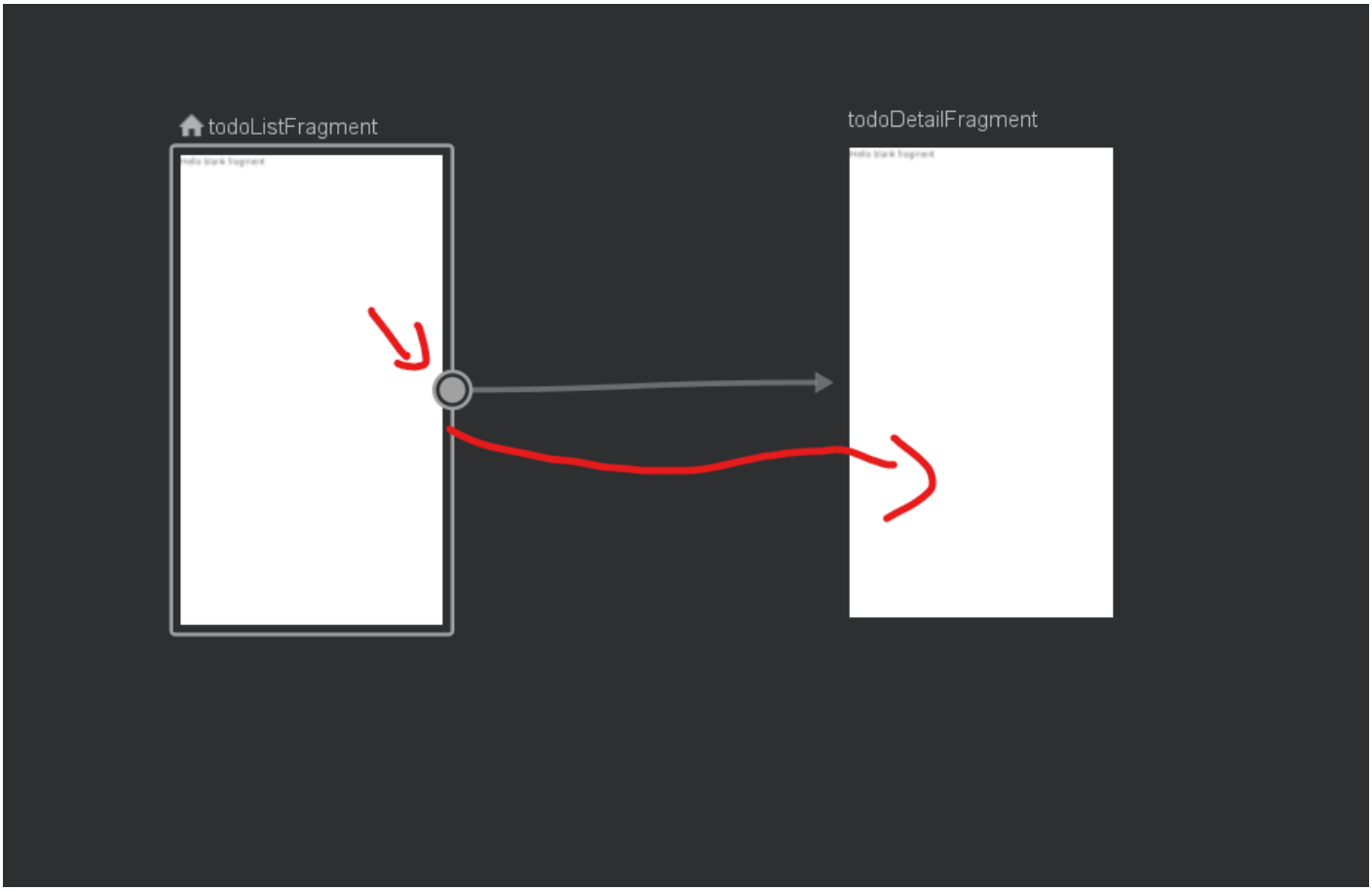


Ainsi, en revenant sur notre Navigation (en rouvrant `res/navigation/todo_navigation.xml`) on peut constaté que notre Activity a bien été ajoutée comme hôte. On peut désormais ajouter nos

fragment à notre navigation :



Et les connecter entre eux par drag & drop :



Vous pouvez séléctionner la flèche et changer son `id` dans l'éditeur graphique par quelquechose de plus court que ce qui a été mis par défaut, par exemple `actionDetail`. Enfin pour activer la génération du code correspondant lancez le Build de votre projet. Cela va nous permettre d'utiliser la navigation dans notre code pour pouvoir effectivement naviguer.

Pour ce faire, ajout un `Button` dans le layout de votre `TodoListFragment` et créez dans son code behing une méthode telle que :

```
public void navigateToDetail(View button) {


}
```

Il faut maintenant lier notre méthode à l'événement de notre bouton : Pour ce faire, donnons un `android:id` à notre `Button` afin de pouvoir la récupérer :

```
<Button
          android:id="@+id/button"
          ....
/>
```

Ensuite on peut récupérer le `Button` à l'aide de la bibliothèque ButterKnife :

```
@BindView(R.id.button)
public Button button;
```

Il faut aussi changer notre `onCreateView` comme suit avant d'activer ButterKnife pour ce Fragment :

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
  View view = inflater.inflate(R.layout.fragment_todo_list, container, false);
  ButterKnife.bind(this,view);
  return view;
}
```

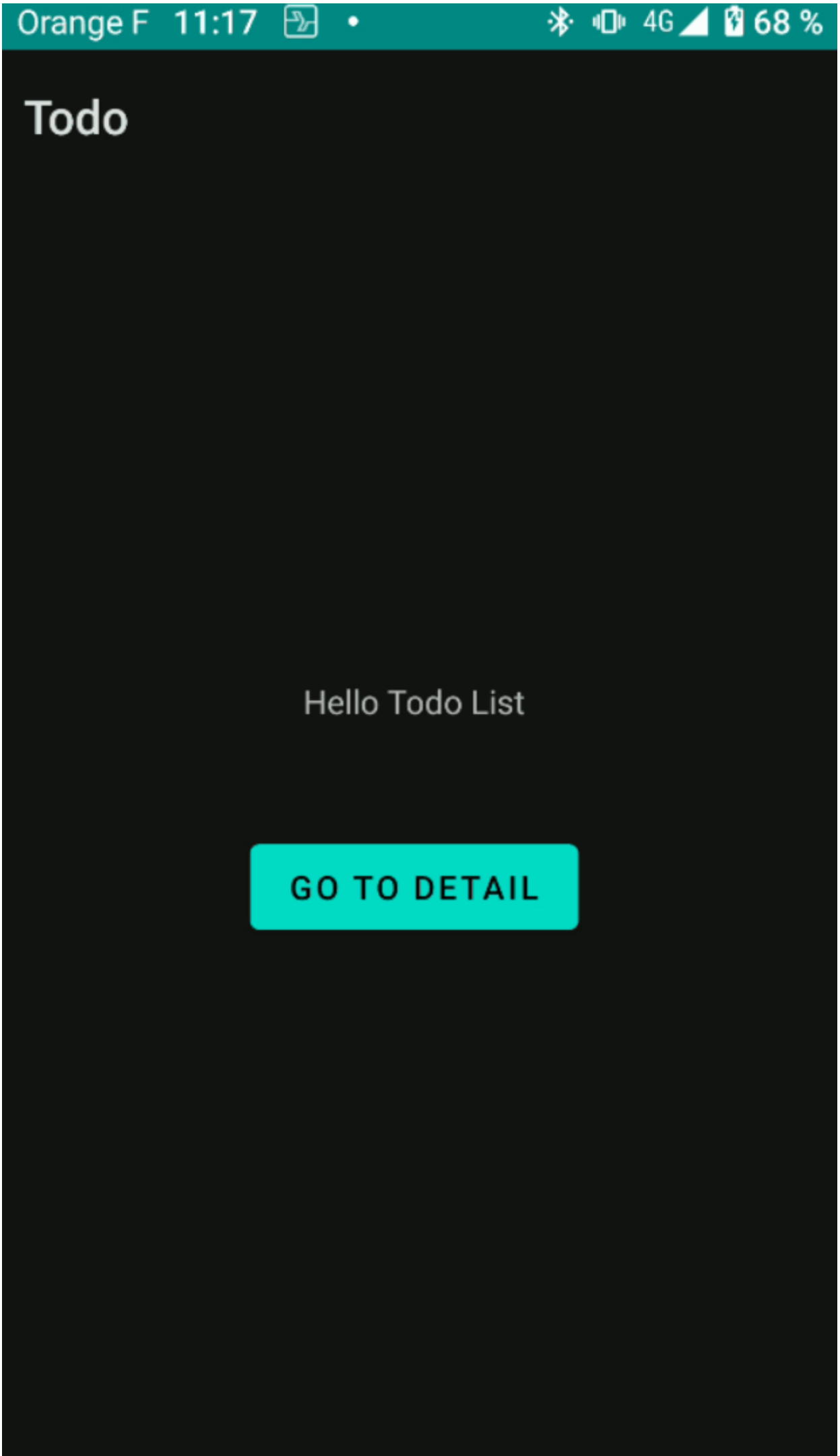
On va maintenant dans notre `TodoDetailFragment` override une nouvelle méthode, `onViewCreated`, qui va être appelée par le framework juste après la création du Fragment. C'est là qu'on va lié notre événement :

```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
  button.setOnClickListener(this::navigateToDetail);
}
```

On peut ensuite rajouter dans la méthode le code de navigation :

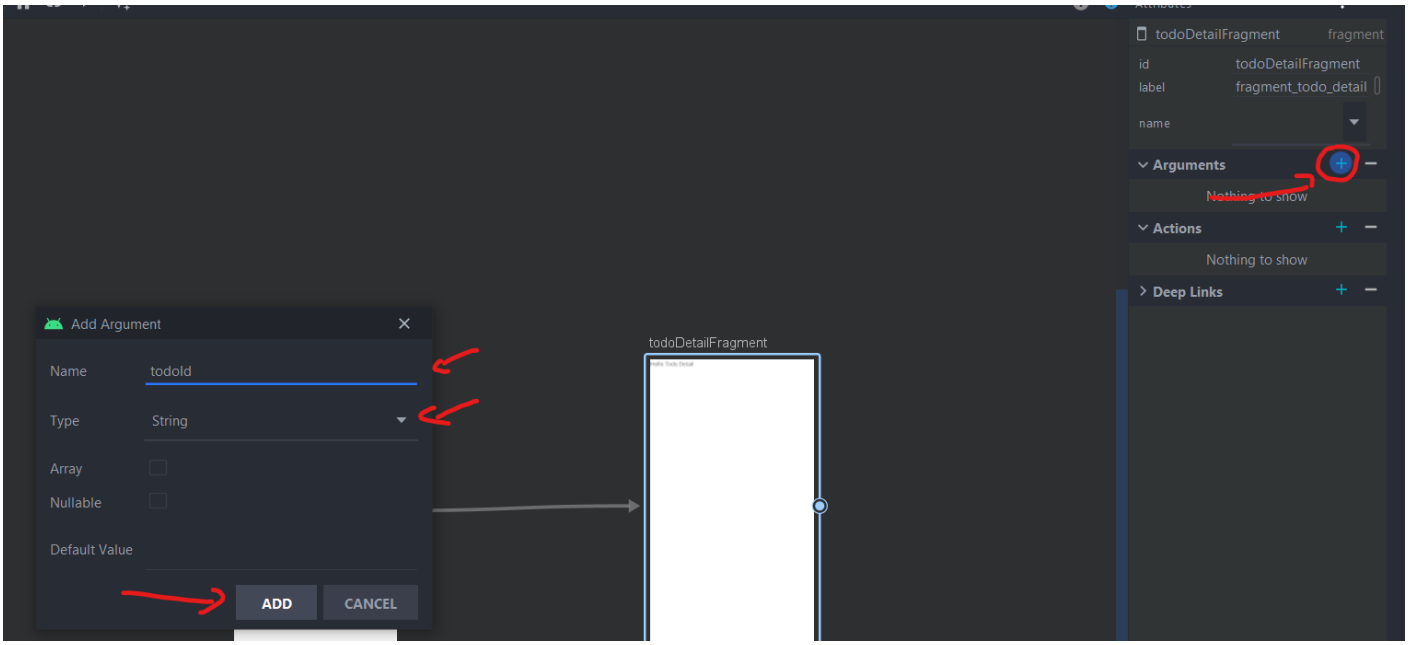
```
Navigation.findNavController(button).navigate(TodoListFragmentDirections.actionDetail());
```

On récupère le controleur de navigation du Framework, et on lui passe l'action de navigation qui a été générée. Vous pouvez maintenant lancer votre app sur un terminal (physique ou émulé) pour tester tout ça.


Orange F 11:17
4G 68 %
Todo
Hello Todo List
GO TO DETAIL

Navigation avec des arguments

Dans l'UI de navigation, en cliquant sur un écran, vous pouvez le configurer. Dans le menu de configuration de votre écran vous pouvez ajouter un argument.



On doit maintenant modifier notre code pour gérer le passage de l'argument, mais d'abord il faut build notre projet pour mettre à jours le code généré. Voici comment rajouter le passage de l'argument, il suffit de le passer en paramètre à notre action :

```
TodoListFragmentDirections.ActionDetail actionDetail =
TodoListFragmentDirections.actionDetail(UUID.randomUUID().toString());
Navigation.findNavController(button).navigate(actionDetail);
```

Nous allons ensuite récupérer et afficher notre UUID dans le DetailFragment. Pour ce faire, donnons un `android:id` à notre `TextView` afin de pouvoir la récupérer :

```
<TextView
          android:id="@+id/detailTextView"
          ....
/>
```

Ensuite on peut récupérer la `TextView` à l'aide de la bibliothèque ButterKnife :

```
@BindView(R.id.detailTextView)
public TextView textView;
```

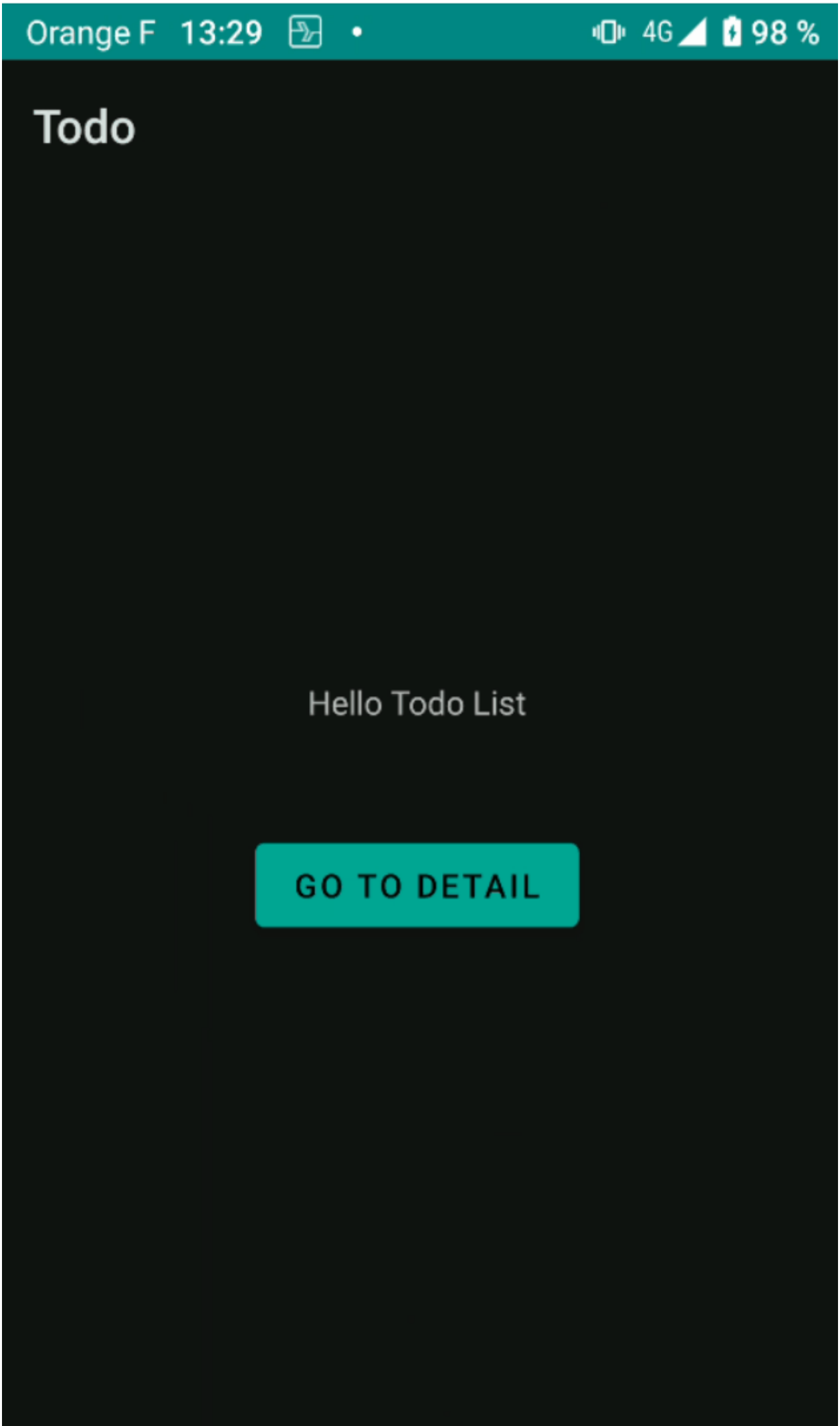
Il faut aussi changer notre `onCreateView` comme suit avant d'activer ButterKnife pour ce Fragment :

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
  View view = inflater.inflate(R.layout.fragment_todo_detail, container, false);
  ButterKnife.bind(this,view);
  return view;
}
```

On va maintenant dans notre `TodoDetailFragment` override une nouvelle méthode, `onViewCreated`, qui va être appelée par le framework juste après la création du Fragment. C'est là qu'on va récupérer notre argument :

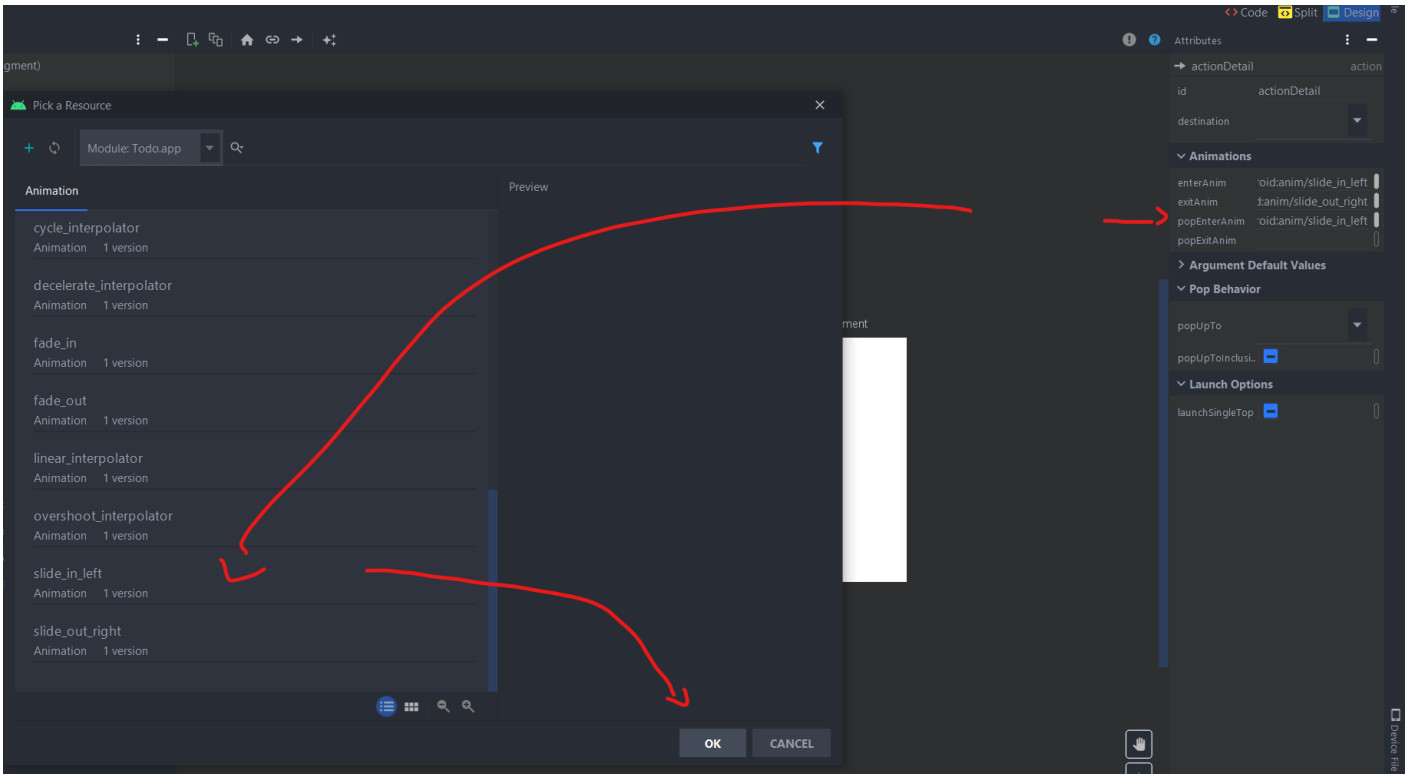
```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
  if(getArguments() != null){
    String uuid = TodoDetailFragmentArgs.fromBundle(getArguments()).getTodoId();
    textView.setText(uuid);
  }
}
```

Résultat :


Orange F 13:29
4G 98 %
Todo
Hello Todo List
GO TO DETAIL

Animations de navigation

Dans l'UI de navigation, en cliquant sur votre action (la flèche) vous pouvez la configurer. Dans le menu de configuration de votre action, vous pouvez configurer des animations. Il y a en a plusieurs proposées par défaut, vous pouvez en choisir et tester le résultat :



Vous pouvez en créer des customisées mais c'est hors de la scope de ce tuto.

Revision #12
Created 2021-01-31 14:42:44 UTC by Arsène Lapostolet
Updated 2021-02-11 10:55:59 UTC by Arsène Lapostolet