Fakher KARCHID

BTS SIO option SLAM

ATELIER 2 - Application Android

Gestionnaire de formations

Rappel du contexte de la mission :

L'application permet de consulter la liste des titres des formations mises à disposition.
L'objectif est de faire évoluer l'application pour gérer les favoris.

Table des matières

Logiciels utilisés :

IDE : Android studio 2021.1.1 Patch 1

SDK version : 32

SGBD : MySQL

Lien BDD : API REST

Mission 1 : Gérer les filtres des formations :

Pour cela il faut créer une méthode dans le contrôleur qui retourne la liste ne contenant que les formations dont le titre contient le filtre reçu en paramètres :

```
/**
 *Filtre pour Favori et Formations
 * @param filtre
 * @param lesFormations
 * @return
 */
public ArrayList<Formation> getLstFiltres(String filtre, ArrayList<Formation> lesFormations)
{
    ArrayList<Formation> lstFiltres = new ArrayList<>();
    for(Formation uneFormation : lesFormations){
        if(uneFormation.getTitle().toUpperCase().contains(filtre.toUpperCase())){
            lstFiltres.add(uneFormation);
        }
    }
    return lstFiltres;
```

Ensuite il faut créer dans la vue de la liste des formations, la méthode événementielle sur le clic du bouton "filtrer" qui doit vérifier si la zone de saisie est vide ou non pour savoir quelle méthode appeler dans le contrôleur, pour valoriser la liste de formations à utiliser pour l'affichage.

```
/**
 * Procedure evenementielle sur le filtre
 */
private void ecouteFiltre(){
    btnFiltrer.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(txtFiltre.getText().toString() != "") {
                lesFormations = controle.getLesFormationFiltre(txtFiltre.getText().toString());
            }
            else{
                lesFormations = controle.getLesFormations();
            }
            creerListe();
        }

    });

}
```

Mission 2 : gérer les favoris

Partie 1 :

Il faut d'abord ajouter l'outils qui nous permet de gérer l'accés à la base de données local sqlite, puis configurer dans le modèle l'accès à la base local et les requêtes à effectuer.

Nous avons besoin de récupérer la liste des favoris de la façon suivante :

```
/**
 * retourne une liste idformation de favori et met à jour la liste des favoris
 * @return dernier profil
 */
public void recupFavori(){
    ArrayList<Integer> listeFavori = new ArrayList<~>();
    bd = accesBD.getReadableDatabase();
    String req = "select * from favori";
    Cursor curseur = bd.rawQuery(req, selectionArgs: null);
    while(curseur.moveToNext()){
        Integer idformation = curseur.getInt( i: 0);
        listeFavori.add(idformation);
    }
    curseur.close();
    controle.setLesFavoris(listeFavori);
}
```

Nous avons aussi besoin d'ajouter l'id d'une formation à la BDD favori :

```
/**
 * ajout d'un formation dans la BDD favori
 * @param idformation
 */
public void ajout(Integer idformation){

    bd = accesBD.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("idformation", idformation);
    bd.insert( table: "favori", nullColumnHack: null, values);
    bd.close();
}
```

Enfin il faut pouvoir supprimer l'id d'une formation de la BDD favori :

```
 */

public void remove(Integer idformation){

    bd = accesBD.getWritableDatabase();
    bd.delete( table: "Favori", whereClause: "idformation=?", new String[]{idformation.toString()});
}
```

Dès le démarrage de l'application, on récupère la liste des favoris en appelant la méthode recupfavori() de la classe accesdistant et on fait le lien avec la liste des formations dans le contrôleur de la façon suivante :

```
public ArrayList<Formation> recupFavoris(){

    if(lstIdFavoris != null) {

        for (Formation uneFormation : lesFormations) {
            for (Integer idformation : lstIdFavoris) {

                if (idformation == uneFormation.getId()) {
                    lstFavoris.add(uneFormation);
                }
            }
        }

    }
    return lstFavoris;
}
```

La liste des favoris ne sera plus jamais récupérée mais sera modifié au fur et à mesure qu'on enlève une formation des favoris ou qu'on en ajoute une :

```
  /**
   * Supprime un favori de la liste est de la table
   * @param unFavori
   */

  public void removeFavori(Formation unFavori)
  {
      accesLocal.remove(unFavori.getId());
      removeLstFavoris(unFavori);

  }
```

```
/**
 * Ajoute un Favori à la table favori et à liste
 * @param unFavori
 */
public void addFavori(Formation unFavori)
{
    accesLocal.ajout(unFavori.getId());
    addLstFavoris(unFavori);

}
```

Il ne reste plus qu'à coder la vue, pour cela on ajoute une méthode évènementielle sur le clic d'un cœur pour déterminer s'il faut ajouter ou supprimer une formation des favoris de la façon suivante :

```
btnListFavori.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        int position = (int)v.getTag();

        if(controle.isFormationFavori(lesFormations.get(position)))
        {
           btnListFavori.setImageResource(R.drawable.coeur_gris);
           controle.removeFavori(lesFormations.get(position));

        }
        else{
            Log.v( tag: "position", msg: "Item: " + position);
            btnListFavori.setImageResource(R.drawable.coeur_rouge);
            controle.addFavori(lesFormations.get(position));


        }

    }
});
    }
notifyDataSetChanged();
```

Il ne faut aussi appeler aussi la fonction notifyDataSetChanged() pour que la vue soit actualisé.

En amont il faut ajouter un setTag() sur l'item qui nous permettra de faire le lien avec la méthode évènementielle :

```
    viewProperties.btnListFavori.setTag(i);

    btnFavoriOnClickListener(viewProperties.btnListFavori);
```

Au moment du premier chargement de la vue il faut pouvoir afficher les favoris ,avec un cœur rouge, et qui ont déjà été stockés dans la base de données avec une condition :

```
    if(controle.isFormationFavori(lesFormations.get(i)))
    {
        viewProperties.btnListFavori.setImageResource(R.drawable.coeur_rouge);

    }
    else{
        viewProperties.btnListFavori.setImageResource(R.drawable.coeur_gris);
    }
```

Partie 2 :

Il faut coder la deuxième vue qui va afficher exclusivement la liste des favoris, le principe est similaire à la première vue, nous devons donc coder une nouvelle activity qui utilisera le même FormationListAdapter .

Mais cette fois-ci il faut valoriser l'adapter avec une liste de favoris et non de formations :

```
// On récupere la liste des favoris pour valoriser l'adapter
lesFormations = controle.getLesFavoris();
controle.setFavoriWindow(true);
```

En amont il faut créer un booléen dans le contrôleur pour pouvoir déterminer si nous sommes dans l'activity des favoris ou dans l'autre. Cela nous permet de mettre une condition sur la méthode évènementielle qui est appelé dans getview() pour pouvoir supprimer des favoris dès qu'on clique sur un cœur rouge :

```
private void btnFavoriOnClickListener(ImageButton btnListFavori){


if(controle.getFavoriWindow()) {
    btnListFavori.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int position = (int) v.getTag();
            controle.removeFavori(lesFormations.get(position));
            notifyDataSetChanged();

        }

    });
}else{
btnListFavori.setOnClickListener(new View.OnClickListener() {
```

Le reste est similaire à la première vue, il faut néanmoins rajouter dans le mainactivity un évènement sur le BtnFavori pour pouvoir accéder à cette activity :

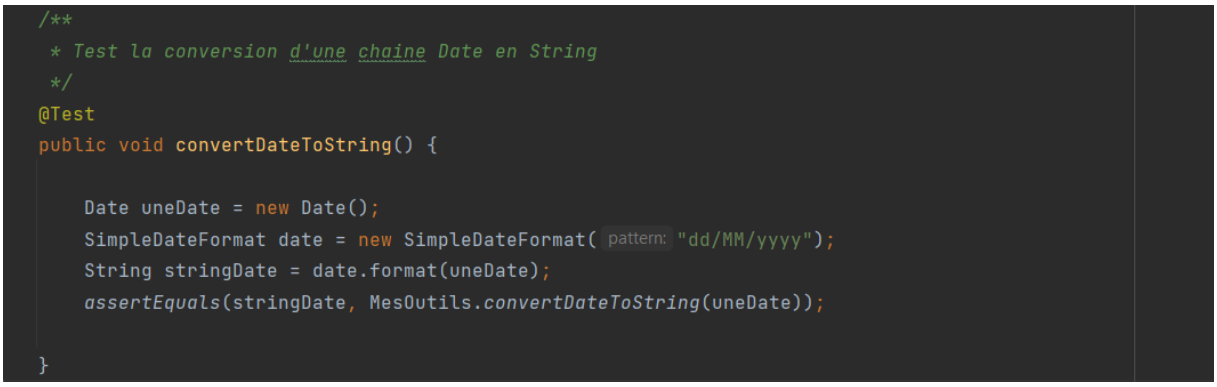
```
private void creerMenu(){
    ecouteMenu((ImageButton)findViewById(R.id.btnFormations), FormationsActivity.class);
    ecouteMenu((ImageButton)findViewById(R.id.btnFavoris), FavoriActivity.class);
}
```

Mission 3 : qualité, test et documentation technique :

Il faut créer des tests unitaires sur les dates :

```java
/**
 * Test la conversion d'une chaine Date en Date
 */
@Test
public void convertStringToDate() {
    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd hh:mm:ss");
    Date date = null;
    try {
        date = formatter.parse( source: "2020-12-28 22:00:29");
    } catch (ParseException e) {
        e.printStackTrace();
    }
    assertEquals(date, MesOutils.convertStringToDate( uneDate: "2020-12-28 22:00:29",  expectedPattern: "yyyy-MM-dd hh:mm:ss"));


}
```



```java
/**
 * Test la conversion d'une chaine Date en String
 */
@Test
public void convertDateToString() {

    Date uneDate = new Date();
    SimpleDateFormat date = new SimpleDateFormat( pattern: "dd/MM/yyyy");
    String stringDate = date.format(uneDate);
    assertEquals(stringDate, MesOutils.convertDateToString(uneDate));

}
```

Et des scenarios de tests, pour cela on crée un scenario pour l'accès à à la liste des formations :

```
@Test
public void scenario(){


    onView(withId(R.id.btnFormations)).perform(click());
    onView(withId(R.id.txtFiltre)).perform(typeText( stringToBeTyped: "Eclipse"), closeSoftKeyboard());
    onView(withId(R.id.btnFiltrer)).perform(click());
    onView(isRoot()).perform(ViewActions.pressBack());
    onView(withId(R.id.btnFormations)).perform(click());
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(2).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(3).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(4).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(5).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(6).onChildView(withId(R.id.btnListFavori)).p
    onView(isRoot()).perform(ViewActions.pressBack());
    onView(withId(R.id.btnFormations)).perform(click());
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(2).onChildView(withId(R.id.btnListFavori)).p
    onView(isRoot()).perform(ViewActions.pressBack());
```

Et un autre scénario pour l'accès à la liste des favoris :

```
@Test
public void scenariot(){

    onView(withId(R.id.btnFavoris)).perform(click());
    onView(withId(R.id.txtFiltre)).perform(typeText( stringToBeTyped: "Eclipse"), closeSoftKeyboard());
    onView(withId(R.id.btnFiltrer)).perform(click());
    onView(isRoot()).perform(ViewActions.pressBack());
    onView(withId(R.id.btnFavoris)).perform(click());
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onView(isRoot()).perform(ViewActions.pressBack());



}
```

Il ne reste plus qu'a générer la documentation technique , l'apk et le déploiement.

Le déploiement à été effectué sur la plateforme amazon aws avec le service EC2 sur une machine amazon-linux après avoir installé Lamp en amont.

Bilan :

L'application nous permet désormais d'ajouter des favoris, de les supprimer, de filtrer les formations et d'accéder à la liste des favoris.

Liste des compétences couvertes :

B1 : Support et mise à disposition de services informatiques

B2 : BLOC 2 – SLAM – Conception et développement d'applications

B3 : SLAM - Cybersécurité des services informatiques – 2e année