Compte-Rendu du code SAE 32

Présentation générale

L'application **Chat-Stream d'entreprise** est une application Android permettant de poster des messages, de leur attribuer des "J'aime", et de les afficher avec une priorité basée sur un système de score d'actualité. Elle utilise **Firebase** comme base de données pour sauvegarder et récupérer les messages. Dans le cadre de mon projet, l'une des étapes fondamentales a été d'établir une connexion entre l'application Android et la base de données Firebase. Cette connexion est essentielle pour stocker et récupérer les données liées à l'authentification des utilisateurs, les messages échangés dans le chat-stream, ainsi que leurs interactions (likes).

Étapes de configuration de Firebase avec Android Studio

Création du projet Firebase

- Je me suis connecté à la console Firebase.
- J'ai créé un nouveau projet Firebase en renseignant le nom du projet et en acceptant les conditions générales d'utilisation.
- Une fois le projet créé, j'ai ajouté une application Android en renseignant le nom du package de l'application (obligatoire pour la connexion). j'ai utilisé com.example.myappli_sae32

Ajout du fichier google-services.json

- Après avoir enregistré l'application Android sur Firebase, j'ai téléchargé le fichier de configuration google-services.json.
- Ce fichier a été placé dans le répertoire app de mon projet Android Studio pour permettre à Firebase d'identifier mon application.

Ajout des plugins / dépendances Firebase

J'ai modifié les fichiers Gradle de l'application pour intégrer Firebase. Voici les changements effectués :

fichier build.gadle.kts(:myappli_sae32) :

- ajout de ces lignes :

```
plugins {
    alias(libs.plugins.android.application) apply false
    id("com.google.gms.google-services") version "4.4.2" apply false

}
```

- Permet d'avoir la bonne version pour pouvoir associer l'application android studio avec FIREBASE.

fichier build.gradle.kts(:app) :

- ajout des dependences :

```
    implementation(platform("com.google.firebase:firebase-bom:33.6.0"))
    implementation("com.google.firebase:firebase-analytics")
}
```

Pour implementation(platform("com.google.firebase:firebase-bom:33.6.0")) :

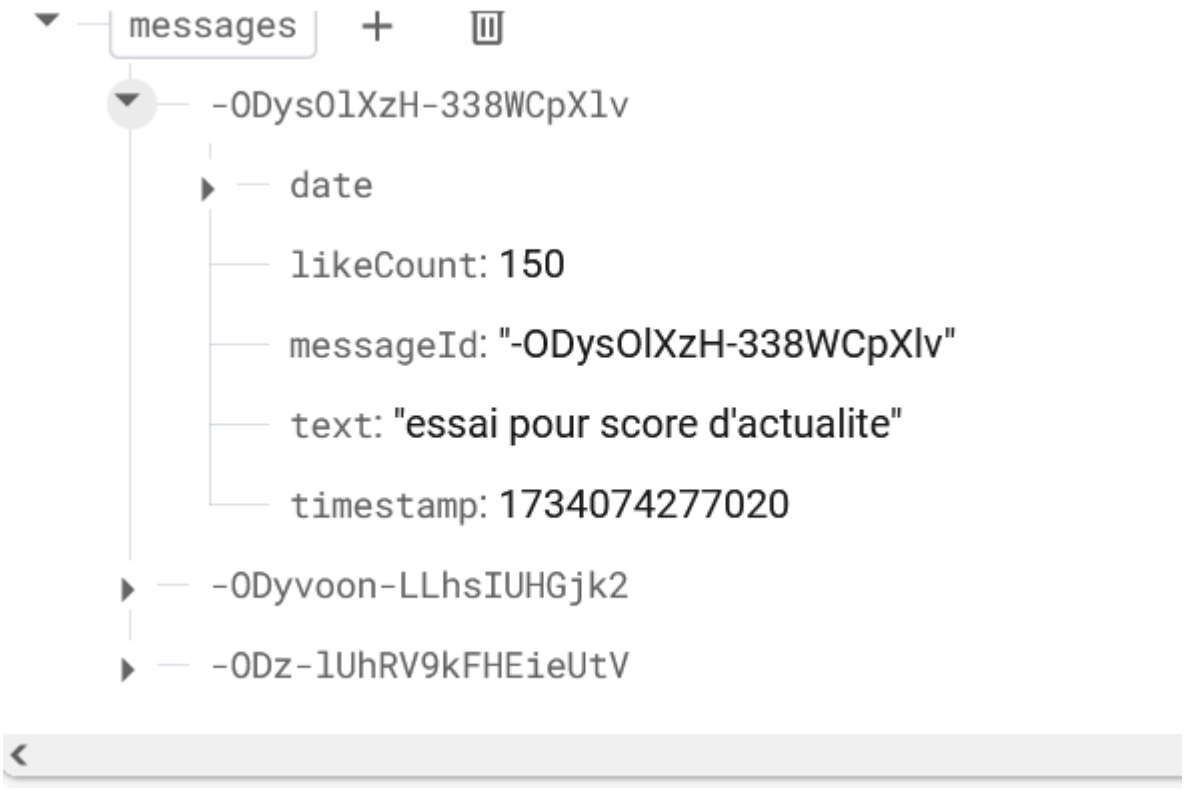
- Cette commande importe la **Firebase BOM (Bill of Materials)**, une fonctionnalité qui simplifie la gestion des versions des bibliothèques Firebase.
- La BOM garantit que toutes les bibliothèques Firebase que vous utilisez dans votre projet Android sont compatibles entre elles.

Pour implementation("com.google.firebase:firebase-analytics"):

- Cette commande ajoute la bibliothèque **Firebase Analytics** au projet Android.
- Firebase Analytics est un outil permettant de collecter et d'analyser les données d'utilisation de l'application. Il permet de comprendre le comportement des utilisateurs et d'améliorer l'application en conséquence.

Ces commandes viennent du protocole de FireBase pour l'association de l'application avec la base de données.

Voici les messages stockées dans ma base de données FireBase :



Une fois cette partie faite, le reste sera à programmer dans le fichier Java pour le stockage des messages, mais pour le moment je vais commencer par la mise en forme de mon application avec le fichier activity_main.xml :

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">
```

Explication commande par défaut :

- Cette ligne spécifie que le document est un fichier XML.
- L'attribut encoding="utf-8" indique que le fichier utilise l'encodage UTF-8, un standard qui prend en charge une large gamme de caractères
- **xmlns:android="http://schemas.android.com/apk/res/android"** : Définit l'espace de noms pour les attributs spécifiques à Android.

Explication pour les ajouts :

- **LinearLayout** : Une disposition qui aligne ses enfants dans une seule direction (verticale ou horizontale).
- **android:layout_width="match_parent"** : La largeur du conteneur s'adapte à celle de l'écran.
- **android:layout_height="match_parent"** : La hauteur du conteneur s'étend également sur tout l'écran.
- **android:orientation="vertical"** : Les éléments enfants seront empilés verticalement.
- **android:padding="16dp"** : Ajoute une marge autour du contenu.

Balise enfants

```
<!-- Titre de l'application -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Chat-Stream d'entreprise"
    android:textSize="24sp"
    android:textStyle="bold"
    android:gravity="center"
    android:layout_gravity="center"
    android:padding="16dp" />
```

Explication :

- TextView : Affiche un texte statique.
- android:layout_width="wrap_content" : La largeur du texte s'ajuste à son contenu.
- android:layout_height="wrap_content" : La hauteur s'ajuste également au contenu.
- android:text="Chat-Stream d'entreprise" : Définit le texte affiché.
- android:textSize="24sp" : Définit la taille du texte à 24sp (taille adaptée aux utilisateurs).
- android:textStyle="bold" : Le texte est en gras.
- android:gravity="center" : Centre le texte à l'intérieur de la vue.
- android:layout_gravity="center" : Centre l'élément dans son parent.
- android:padding="16dp" : Ajoute une marge autour du texte.

```
<!-- Liste des messages -->
<LinearLayout
    android:id="@+id/messageContainer"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:orientation="vertical" />
```

Explication :

- Une disposition horizontale pour aligner les champs d'entrée et le bouton d'envoi côte à côte.

```
<!-- Champ de texte pour entrer le message -->
<EditText
    android:id="@+id/messageInput"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:hint="Entrez votre message"
    android:padding="10dp"
    android:textSize="16sp" />
```

Explication :

- **EditText** : Un champ de texte où l'utilisateur peut saisir du texte.
- **android:id="@+id/messageInput"** : Identifiant unique pour référencer ce champ dans le code.
- **android:layout_width="0dp"** : La largeur est initialement définie à zéro, mais...
- **android:layout_weight="1"** : Le champ occupe tout l'espace disponible dans la disposition horizontale.
- **android:hint="Entrez votre message"** : Affiche un texte d'indication lorsque le champ est vide.
- **android:padding="10dp"** : Ajoute une marge au champ.
- **android:textSize="16sp"** : Définit la taille du texte.

```
<!-- Bouton pour envoyer le message -->
<Button
    android:id="@+id/sendButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Envoyer"
    android:layout_marginStart="8dp"
    android:textSize="16sp" />
LinearLayout>
```

Explication :

- **Button** : Un bouton cliquable.
- **android:id="@+id/sendButton"** : Identifiant unique pour référencer le bouton.
- **android:layout_width="wrap_content"** : La largeur du bouton s'ajuste à son contenu.
- **android:layout_height="wrap_content"** : La hauteur s'ajuste également au contenu.
- **android:text="Envoyer"** : Texte affiché sur le bouton.
- **android:layout_marginStart="8dp"** : Ajoute une marge entre le bouton et le champ de texte.
- **android:textSize="16sp"** : Taille du texte du bouton.

Une fois la mis en page réalisé je passe au codage sur le fichier Main_Activity pour la fonctionnalité de l'application :

Packages Importés

- **android.os.Bundle** : Permet de gérer l'état d'une activité dans Android, essentiel pour la sauvegarde et la récupération des données.
- **android.view.View** : Permet de gérer l'interaction avec les composants de l'interface utilisateur, comme les clics de bouton.
- **android.widget.Button, EditText, LinearLayout, TextView, Toast** : Ces classes sont utilisées pour créer et gérer les composants de l'interface utilisateur (boutons, champs de saisie, conteneurs de messages, affichage de texte, messages temporaires).

- **com.google.firebase.database.DatabaseReference, FirebaseDatabase, DataSnapshot, DatabaseError, ValueEventListener** : Ces classes sont utilisées pour interagir avec Firebase, permettant de lire et d'écrire des données dans la base de données en temps réel.
- **java.text.SimpleDateFormat, java.util.Date, java.util.Locale** : Ces classes sont utilisées pour manipuler et formater les dates.

onCreate(Bundle savedInstanceState)

Commande :
Cette méthode est appelée au lancement de l'application. Elle initialise l'activité et configure l'interface utilisateur.

Détails :

- **setContentView(R.layout.activity_main) :** Lie cette activité au fichier XML de mise en page activity_main. Ce fichier contient la définition de l'interface utilisateur (UI), comme les boutons et champs de texte.
- **findViewById :** Associe les vues définies dans l'interface utilisateur XML aux variables Java. Exemple :
    - messageInput → Référence au champ de saisie de texte.
    - sendButton → Référence au bouton d'envoi.
    - messageContainer → Référence au conteneur qui affichera les messages envoyés.

sendButton.setOnClickListener(View.OnClickListener)

Commande :
Ajoute une action à effectuer lorsqu'on clique sur le bouton "Envoyer".

Détails :

- **messageInput.getText().toString().trim() :** Récupère le texte saisi dans le champ EditText et supprime les espaces au début et à la fin.
- **Vérification de texte vide :** Si le texte est vide, affiche une notification via un **Toast** (petit message temporaire).
- **Création d'un message :** Si le texte est valide :
    - Crée un nouvel objet Message avec le texte et la date actuelle.
    - Génère un ID unique pour le message via databaseReference.push().getKey().
    - Ajoute le message à Firebase sous cet ID avec databaseReference.child(messageId).setValue(newMessage).
    - Ajoute le message à la liste locale messages.

- Vide le champ de saisie (messageInput.setText("")).
- Met à jour l'affichage des messages avec refreshUI().

addMessageToUI(Message message)

Commande :
Ajoute un message individuel à l'interface utilisateur.

Détails :

- Crée dynamiquement un conteneur (LinearLayout) pour le message.
- Ajoute deux composants dans ce conteneur :
  1. **Texte du message :**
     - Contient le texte, la date et le nombre de likes.
     - Formaté avec formatMessage(Message message).
  2. **Bouton "J'aime" :**
     - Ajoute un bouton qui, une fois cliqué, incrémente le compteur de likes avec message.incrementLike().
     - Met à jour la base de données Firebase avec databaseReference.child(messageId).child("likeCount").setValue(message.getLikeCount()).
- Ajoute ce conteneur au messageContainer principal de l'interface.

refreshUI()

Commande :
Met à jour l'affichage des messages en les triant par score d'actualité.

Détails :

- Supprime tous les messages actuellement affichés via messageContainer.removeAllViews().
- Trie les messages dans messages selon leur score d'actualité :
  - **Formule du score :** score=(30−δ)×α\text{score} = (30 - \delta) \times \alphascore=(30−δ)×α, où :
    - δ\deltaδ est l'âge du message en jours.
    - α\alphaα est le nombre de "likes".
    - Les messages de plus de 30 jours ont un score minimal pour apparaître en bas.
- Ajoute chaque message trié à l'interface via addMessageToUI().

onStart()

Commande :
Appelée lorsque l'activité devient visible. Elle synchronise les données avec Firebase.

Détails :

- Ajoute un **listener** Firebase via databaseReference.addValueEventListener(ValueEventListener) pour surveiller les changements dans la base de données.
- **onDataChange(DataSnapshot dataSnapshot) :**
    - Récupère les messages depuis Firebase.
    - Les convertit en objets Message et les ajoute à la liste locale messages.
    - Appelle refreshUI() pour mettre à jour l'interface utilisateur.
- **onCancelled(DatabaseError databaseError) :** Gère les erreurs éventuelles en affichant un message d'erreur avec un Toast.

6. formatMessage(Message message)

Commande :
Formate le texte d'un message avant de l'afficher.

Détails :

- Utilise SimpleDateFormat pour convertir le timestamp en une date lisible (format : dd/MM/yyyy HH:mm).
- Retourne une chaîne contenant :
    - Le texte du message.
    - La date de création.
    - Le nombre de likes.

calculateMessageScore(Message message, long currentTime)

Commande :
Calcule le score d'actualité d'un message en fonction de sa date et de ses likes.

Détails :

- Calcule l'âge (δ\deltaδ) du message en jours : (currentTime−timestamp)/(1000×60×60×24)(\text{currentTime} -

\text{timestamp}) / (1000 \times 60 \times 60 \times 24)(currentTime−timestamp)/(1000×60×60×24).
- Récupère le nombre de likes (α\alphaα).
- Applique la formule de score.
- Retourne un score très faible pour les messages de plus de 30 jours.

Classe interne Message

Commande :
Définit un modèle pour représenter un message.

Détails :

- **Attributs :**
  - text : Contient le texte du message.
  - likeCount : Compteur de likes, initialisé à 0.
  - timestamp : Heure de création du message, stockée en millisecondes.
  - messageId : ID unique du message dans Firebase.
- **Constructeurs :**
  - Sans argument : Nécessaire pour Firebase.
  - Avec texte et date : Initialise un nouveau message.
- **Méthodes :**
  - incrementLike() : Incrémente le compteur de likes.
  - getDate() : Convertit le timestamp en objet Date.

Firebase Base de données

Commande :
Le code utilise Firebase pour stocker et récupérer les messages.

Détails :

- **Référence Firebase :**
  - databaseReference = FirebaseDatabase.getInstance().getReference("messages"); établit un lien avec la collection "messages".
- **Écriture dans Firebase :**
  - databaseReference.child(messageId).setValue(newMessage) ajoute un message sous un ID unique.
  - databaseReference.child(messageId).child("likeCount").setValue(message.getLikeCount()) met à jour le compteur de likes.

- **Lecture depuis Firebase :**
    - databaseReference.addValueEventListener(ValueEventListener) écoute les modifications en temps réel.

Rendu final de l’application :