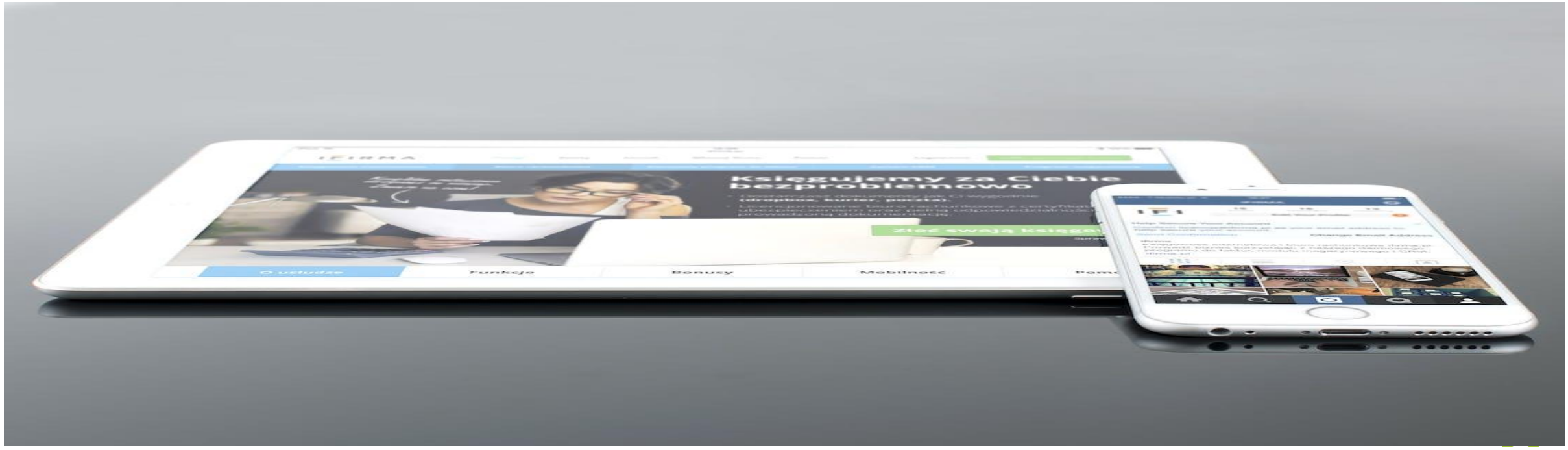
Architecture d'Application

ENSEIGNANTE: MIRNA AWAD

RESPONSABLE DE COURS: A. Toudeft



PLAN

- Introduction
- Architecture MVVM
  - Le Modèle
  - La vue
  - Le ViewModel
- Mise en oeuvre sous Android
- Exemple de démonstration
- Exercice

INTRODUCTION

- La mise en place d'une bonne architecture permet :
  - De faciliter l'évolution de l'application (ajout de fonctionnalités, amélioration, …);
  - De bien répartir les responsabilités entre les différents composants de l'application;
  - De faciliter le test des composantes de l'application;
  - De faciliter la réutilisation;
  - D'éviter la redondance du code.
- Pattern standard : MVC (Modèle-Vue-Contrôleur);
- MVP (Model-View-Presenter) et MVVM (Model-View-ViewModel) sont très utilisées dans le mobile.

ARCHITECTURE MVVM

- Modèle-Vue-ViewModel *(Model-View-ViewModel)*;
- Modèle architectural couramment utilisé en mobile Android;
- Séparation claire des responsabilités => Facilite le test, la maintenance et l'évolutivité de l'application;
- 3 parties :
  - Modèle;
  - Vue;
  - ViewModel.

MODÈLE/MODEL

- Représente les données de l'application;
- Responsable de la récupération, de la manipulation et de la gestion des données;
- Aucune logique liée à l'interface utilisateur.

VUE/VIEW

- Représente l'interface utilisateur de l'application;
- Aucune logique métier;
- Se contente d'observer les données exposées par le ViewModel et de mettre à jour l'interface;
- Ne communique pas directement avec le Modèle.

VIEWMODEL

- Intermédiaire entre le modèle et la vue;
- Récupère les données du modèle et les met en forme pour que la vue les affiche;
- Utilise LiveData pour répercuter les changements en temps réel et notifier la Vue.
- Répond aux actions de l'utilisateur et effectue les traitements nécessaires sur le modèle.

MISE EN OEUVRE SOUS ANDROID - VUE

- Activités et/ou fragments;
- Affiche les composants de l'interface;
- Réagit aux actions de l'utilisateur en gérant les événements associés (*onClick()*,...);
- Détient une/des références vers le ou les ViewModel(s) dont elle a besoin, et **observera** les changements des données.
- L'observation reste active tant que l'Activity est en vie (c'est-à-dire, tant qu'elle n'est pas détruite).

MISE EN OEUVRE SOUS ANDROID - VUE

Exemple – Dans MainActivity :

- On récupère une instance de CompteBancaireViewModel via ViewModelProvider.

```
CompteBancaireViewModel viewModel = new
ViewModelProvider(this).get(CompteBancaireViewModel.class);
```

- On observe les comptes bancaires (LiveData) de viewModel

```
viewModel.getComptes().observe(this, new Observer<List<CompteBancaire>>() {
            @Override
            public void onChanged(List<CompteBancaire> comptes) {
                // actions automatiques lorsque les données changent
            }
});
```

MISE EN OEUVRE SOUS ANDROID - VIEWMODEL

- Une ou plusieurs classes qui héritent de la classe **ViewModel** ou **AndroidViewModel**.
- Contient la logique métier de l'application;
- Interagir avec le modèle pour récupérer et manipuler les données;
- Exposer les données sous forme de LiveData pour que la Vue les observe.

MISE EN OEUVRE SOUS ANDROID - VIEWMODEL

- **ViewModel** vs. **AndroidViewModel**:
  - ViewModel: Ne contient aucune référence au contexte
  - AndroidViewModel: Étend ViewModel en fournissant l'objet Application, permettant l'accès au contexte.

- **MutableLiveData <T>** vs. **LiveData <T> :**
  - MutableLiveData<T>**:** MutableLiveData nous permet de mettre à jour les données de manière dynamique*. (write access)*
  - LiveData<T> **:** exposez uniquement les données à l'interface utilisateur afin que les vues puissent les observer mais ne puissent pas les modifier. *(Read-only access)*

MISE EN OEUVRE SOUS ANDROID - VIEWMODEL

- Exemple:

```
public class CompteBancaireViewModel extends ViewModel {
    private MutableLiveData <List<CompteBancaire>> comptes = new MutableLiveData<>();

    public LiveData <List<CompteBancaire>> getComptes() {
        return comptes;
    }
    public void chargerComptes() {
          new Thread(() -> {
              List<CompteBancaire> liste = CompteBancaireDao.getComptes();
              comptes.postValue(liste);
          }).start();
      }
  }
```

setValue() doit être appelé sur le thread principal et met à jour la valeur du LiveData de manière synchrone.
postValue() peut être appelé depuis un thread secondaire et planifie la mise à jour du LiveData sur le thread principal de manière asynchrone.

MISE EN OEUVRE SOUS ANDROID - MODÈLE

- Une ou plusieurs classes;
- Représente les données de l'application et fournit des méthodes pour les manipuler.
- Inclut les classes de données (DAO), de gestionnaires de bases de données, de services réseau, les entités etc…

MVVM AVEC REPOSITORY PATTERN

- Ajout d'un **Repository** pour séparer l'accès aux données (API, BD locale, Cache, etc.).
- Permet une meilleure organisation en déléguant la récupération des données à une classe spécialisée (ex: CompteRepository).
- ViewModel interagit uniquement avec le Repository, qui se charge de récupérer les données via Dao de l'API ou de la base de données.

```
public class CompteRepository {
    …
    public void fetchComptesFromAPI(FetchComptesCallback callback) {
        new Thread(() -> {
            try {
                List<CompteBancaire> comptes = CompteBancaireDao.getComptes();
                callback.onFetch(comptes);
            } catch (IOException | JSONException e) {
                callback.onError("Erreur : " + e.getMessage());
            }
        }).start();
    }
}
```

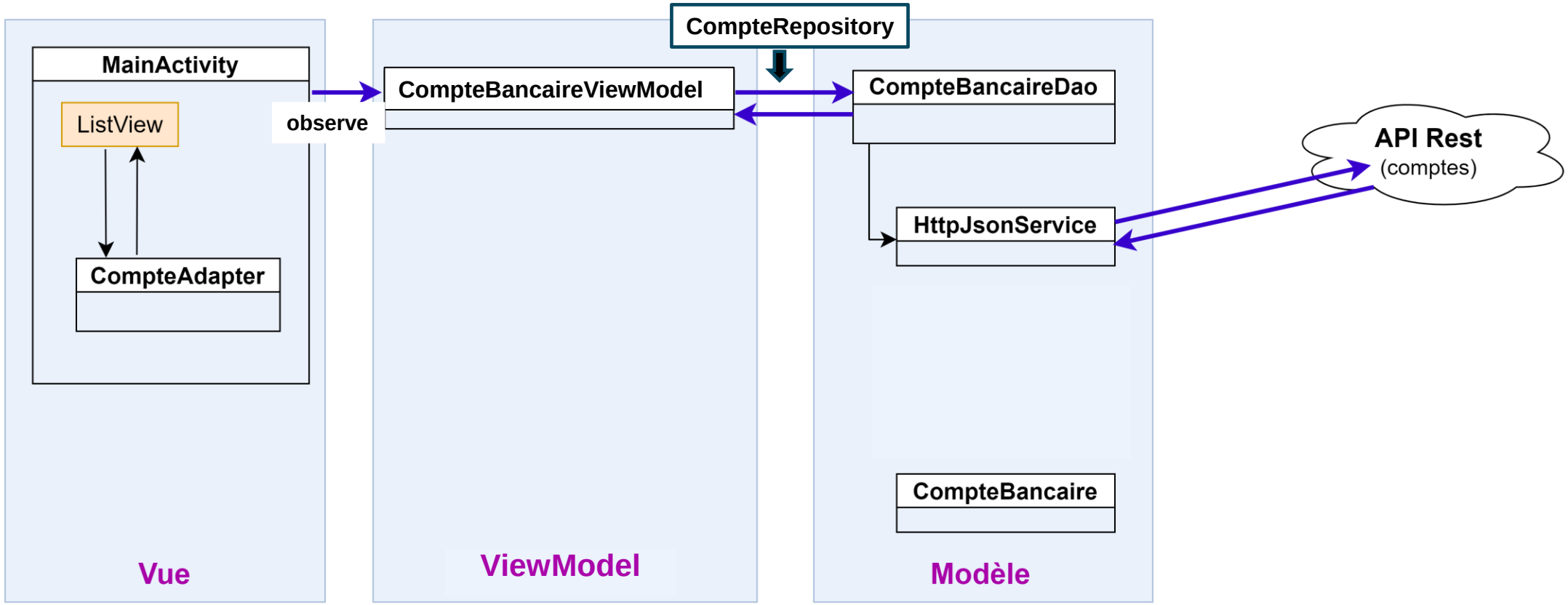
```
public interface FetchComptesCallback {
    void onFetch(List<CompteBancaire> comptes);
    void onError(String errorMessage);
}
```

MVVM AVEC REPOSITORY PATTERN

```
public class CompteBancaireViewModel extends ViewModel {
    private CompteRepository repository;
    private MutableLiveData<List<CompteBancaire>> comptes = new MutableLiveData<>();
    public CompteBancaireViewModel() {
        this.repository = new CompteRepository();
    }
     public void chargerComptes() {
        repository.fetchComptesFromAPI(new CompteRepository.FetchComptesCallback() {
            @Override
            public void onFetch(List<CompteBancaire> comptes) {
                comptes.postValue(comptes);
            }
            @Override
            public void onError(String errorMessage) {
                //message d'erreur
            }});
    }
}
```

DÉMONSTRATION DE L'EXEMPLE DU COURS

- Exemple récapitulatif : Voir projet **Exemple_Architecture_MVVM;**
- C'est le même que **Exercice_Serveur_JSON_Banque** de cours précédent mais organisé selon MVVM.



DÉMONSTRATION DE L'EXEMPLE DU COURS

- Exemple récapitulatif : Voir projet **Exemple_Archi_MVP;**
- Exemple de flux:

MainActivity (affiche les données dans un ListView)

↓ (Observe les données en temps réel)

CompteBancaireViewModel (Gère la logique)

↓ (Appelle le repository)

CompteRepository (Récupère les données depuis les Modèles)

↓

Les Modèles (CompteBancaireDAO, HttpJsonService) (implémente les requêtes API et/ou SQLite)

EXERCICE

- Réaliser l'exercice du cours (fichier ***exercice-cours.pdf***) afin d'ajouter à l'exemple du cours la fonctionnalité permettant d'effectuer des operations de credit et de debit sur les comptes et de mettre à jour les comptes dans le serveur.