ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS
INGENIERÍA EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

PERÍODO ACADÉMICO: 2019B
ASIGNATURA: SIC 634 - **INGENIERÍA DE SOFTWARE II**
PROFESOR: Maritzol Tenemaza
TEMA: Exposición VIPER
FECHA DE ENTREGA: 28-10-2019

AUTORES:

- Patrick Cabezas
- Danny Díaz
- Juan López
- Juan José Morales
- Andrés Pantoja

VIPER

VIPER es una arquitectura Clean para aplicaciones iOs. Se nombra así debido a que divide principalmente en View – Interactor – Presenter – Entity – Routing, las cuales son las palabras que conforman el acrónimo VIPER. Esta división facilita aislar las dependencias y probar las interacciones entre capas.

Surge como alternativa al patrón establecido por MCV (Model – View - Controller) debido a que, en este toda la lógica del negocio se encontraba en el controlador. VIPER se considera como una evolución del MCV, debido a que resulta más escalable, posee una ubicación más clara y mejor distribuida de la lógica del negocio y también permite que muchos desarrolladores trabajen en el proyecto de manera simultánea.

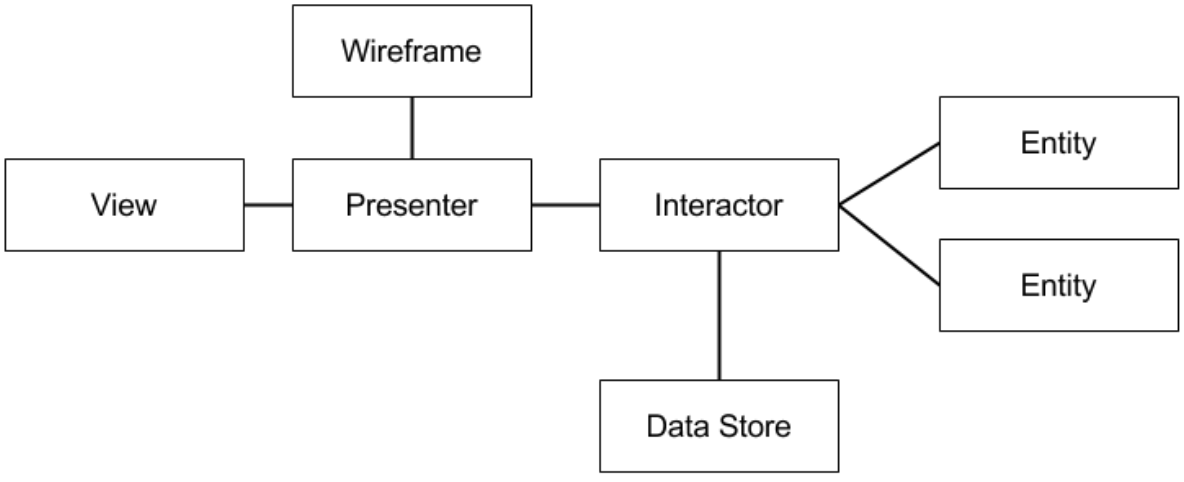

Figura 1, interacción de cada capa de VIPER

VIPER se divide en:

View: Transmite la entrada del usuario al presenter y espera que este le devuelva el contenido que se debe mostrar, la vista nunca pregunta por los datos que va a utilizar. Posteriormente muestra el contenido indicado por el presenter. También le informa al presenter de los eventos que suceden para que tome las decisiones necesarias ante dichos eventos.

Interactor: Contiene la lógica del negocio según las especificaciones recogidas en los casos de uso. Extrae datos desde la capa de model y su implementación es totalmente independiente de la interfaz de usuario. El interactor únicamente puede comunicarse con el DataManager para así obtener Entidades o modelos de negocio que pasaran al presenter y este a su vez los procese hacia la vista.

Presenter: Contiene la lógica de las vistas y recibe eventos desde las mismas, permitiendo la visualización y el control de las entradas. Su relación con el interactor se basa en las acciones del usuario. También genera la vista requerida y la envía al View para que posteriormente sea mostrada.

Entity: Contiene los modelos de objetos básicos que son utilizados por el interactor.

Router: También se conoce como Wireframe. Este contiene la lógica de navegación para describir las pantallas que se muestran y mantener el orden de las mismas. Su importancia radica en que es el punto de entrada a cada uno de los módulos presentes en la app.

Ejemplo:

VIPER es una arquitectura inicialmente hecha para IOS, sin embargo, también puede aplicarse a otro entorno móvil que comparta el flujo de pantallas por medio de “Actividades”. Una actividad en términos de Android y IOS es la pantalla y toda la estructura lógica que contiene.

El proyecto puede encontrarse en https://github.com/HauLeCong/Android-VIPER-Example y similar a la Figura 1 en la Figura 3 se puede observar como interactúa los diferentes elementos de la misma.
Ejecutándo la aplicación en Android Studio se ve como en la imagen de la derecha

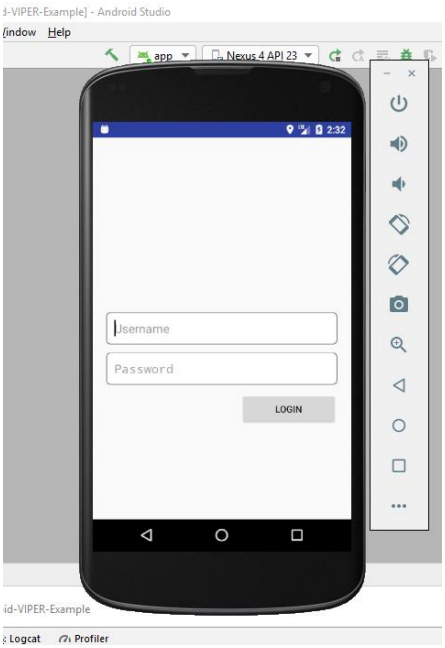

Figura 2, ejecución del programa

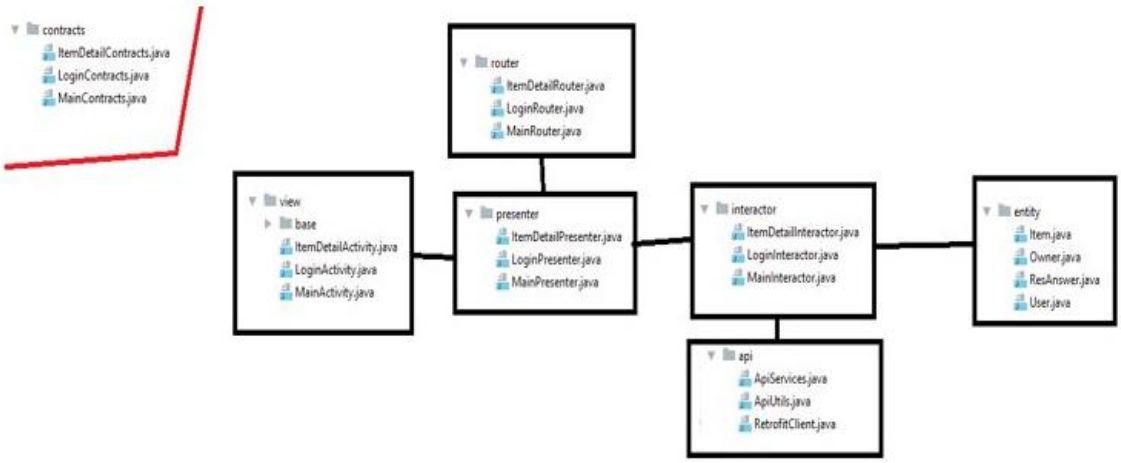

Figura 3, arquitectura de la aplicación Android usando VIPER

Flujo del programa

Primero, la aplicación inicia en la actividad de Login (vista) en el que al presionar el botón de 'login' envía los datos al presentador

```
@OnClick(R.id.btn_login)
public void loginButtonClicked(){
    presenter.onLoginButtonPressed(edtUsername.getText().toString(), edtPassword.getText().toString());
}
```

Y el presentador a su vez envía información al interactor

```
@Override
public void onLoginButtonPressed(String username, String password) {
    interactor.login(username, password);
}
```

El interactor posee toda la lógica del negocio y también la información de las bases de datos con instancias de clases, por lo que al recibir el usuario y la contraseña las valida

```
@Override
public void login(String username, String password) {
    User user = new User();
    user.setUsername("Hau le");
    user.setPassword("123456");

    if(username.equals(user.getUsername()) && password.equals(user.getPassword())){
        output.onLoginSuccess(user);
    }else{
        output.onLoginFailed("Login Failed!!!!!!!!");
    }
}
```

Es importante recalcar que el interactor ya interactúa con las entidades y la base de datos (carpeta de Api y la librería Retrofit). Además, la variable "output" es una referencia al presentador con el que se acaba de comunicar.

Por lo que, si el inicio de sesión es válido, llama a la función "onLoginSuccess" del presentador

```
@Override
public void onLoginSuccess(User user) { router.presentHomeScreen(user); }
```

Y el presentador al recibir del interactor que el inicio de sesión fue válido llama al "router" que en este caso representaría al Wireframe. El "router" se encarga de la navegabilidad entre pantallas, de este modo se da por terminado la actividad de "Login" y se navega hacia otra actividad.

```
@Override
public void presentHomeScreen(User user) {
    Intent intent = new Intent(context, MainActivity.class);
    intent.putExtra( name: "User", user);
    context.startActivity(intent);
    ((Activity)context).finish();
}
```

Nota: Intent es un método que permite mostrar una nueva pantalla poniendo la anterior pantalla en "espera" (almacenada en caché). Aunque la función termina llamando a una **interface** para terminar la actividad por lo que ya no quedará en "espera" sino que se detendrá.

El siguiente código corresponde a la creación de la **vista** MainActivity, por lo que este primer intercambio de mensajes para cambiar de pantalla habrá finalizado.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    initDataView();
}

private void initDataView() {
    presenter = new MainPresenter( view: this);
    //call api
    presenter.actionLoadListAnswer();
    list = new ArrayList<>();
    adapter = new AnswerAdapter( context: this, list, (view, position) -> {
            //TODO: callback onclick item
            presenter.actionItemClick(list.get(position));
    });
    recyclerView.setAdapter(adapter);
}
```

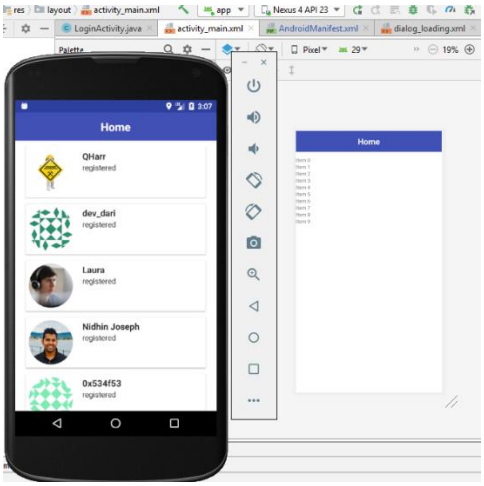

Figura 4, Fin de la actividad Login e inicio de la actividad Main

En la Figura 3, en la parte superior izquierda se puede apreciar una carpeta llamada “contracts”. En esta carpeta se reúne todas las interfaces de cada VIPER de cada actividad, por ejemplo para la actividad del Login se posee las siguientes 5 interfaces. Estas permiten realizar todo el intercambio de información de manera ágil:

```
public interface LoginContracts {
    interface View{
        void showError(String message);
    }

    interface Presenter{
        void onLoginButtonPressed(String username, String password);
    }

    interface Interactor{
        void login(String username, String password);
    }

    interface InteractorOutput{
        void onLoginSuccess(User user);
        void onLoginFailed(String message);
    }

    interface Router{
        void presentHomeScreen(User user);
    }
}
```

Conclusiones:

- El uso de VIPER permite la escalabilidad del proyecto y el trabajo simultáneo de los desarrolladores de manera eficiente.
- Se basa en la división de la aplicación en componentes en función de su rol, por lo que el código fuente resulta más claro, compacto y reusable
- Se puede aplicar para otras plataformas como Android, no solo para IOS.

Recomendaciones:

- Se recomienda el uso de VIPER en proyectos grandes, debido a que disminuye considerablemente su complejidad, al dividirlo en módulos.
- Resulta muy útil cuando es necesario un código con alto desacoplamiento, facilitando la reutilización y el testeo.
- Cuando se necesita controlar de manera eficiente los conflictos, VIPER es una buena opción debido a que es perfecto para la división de tareas al basarse en los principios de responsabilidad única.

Bibliografía:

- Apiumhub. Novoseltseva Ekaterina [Online: Consultado 26/10/2019] https://apiumhub.com/es/tech-blog-barcelona/arquitectura-viper/
- Guía de programación iOS- VIPER (Clean Architecture) [Online: Consultado 26/10/2019] http://www.zeusappsmobile.com/guia-programacion-ios/viper-clean-architecture/