Model-View-Presenter

Стандартные паттерны

1) MVC (Model-View-Controller)

2) MVP (Model-View-Presenter)

3) MVVM (Model-View-ViewModel)

Кратко говоря – Model-View-Delegate

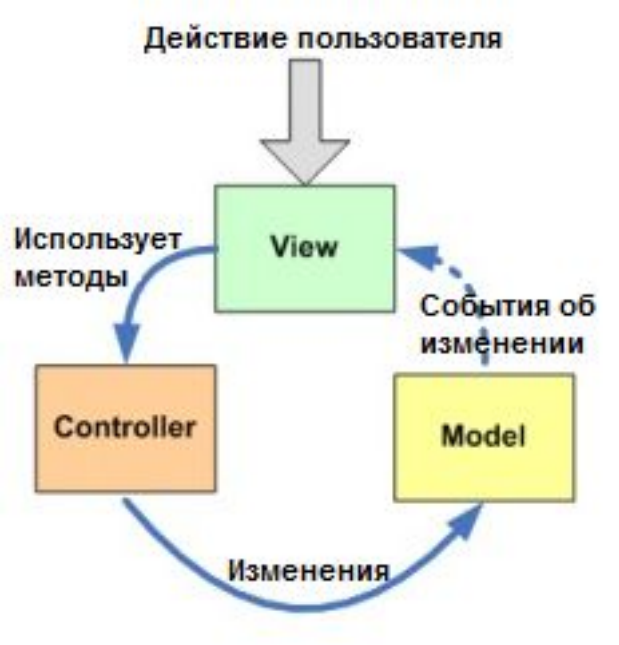
Model

Модельки объектов для взаимодействия View и Presenter

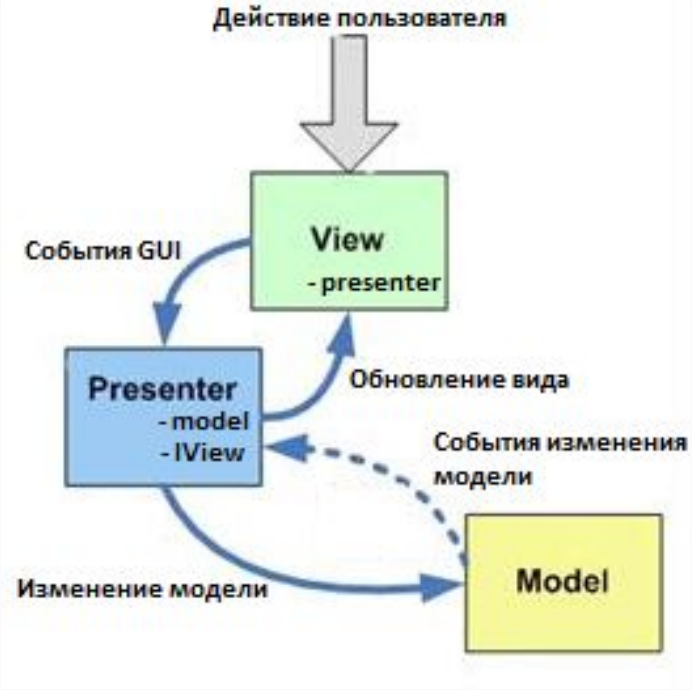
View

1) Отображение данных, полученных от делегата
2) Передает действия пользователя в делегат
3) Управляется делегатом
4) Не содержит логики

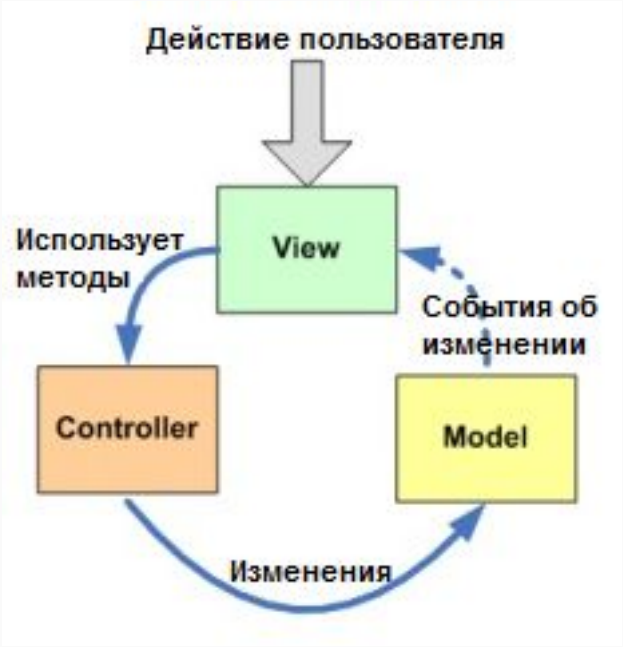
MVC

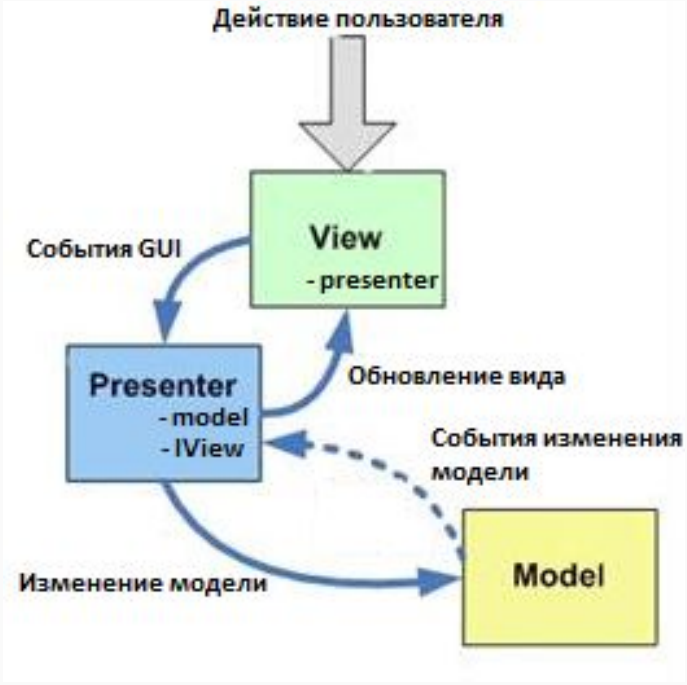

Действие пользователя
View
Использует
методы
События об
изменении
Controller
Model
Изменения

MVP


Действие пользователя
View
- presenter
События GUI
Presenter
- model
- IView
Обновление вида
События изменения модели
Model
Изменение модели

MVC vs MVP


Действие пользователя
View
Использует методы
События об изменении
Controller
Model
Изменения


Действие пользователя
View
- presenter
События GUI
Presenter
- model
- IView
Обновление вида
События изменения модели
Изменение модели
Model

MVC vs MVP

1) Presenter управляет View через интерфейс непосредственно, Controller управляет View опосредованно через Model

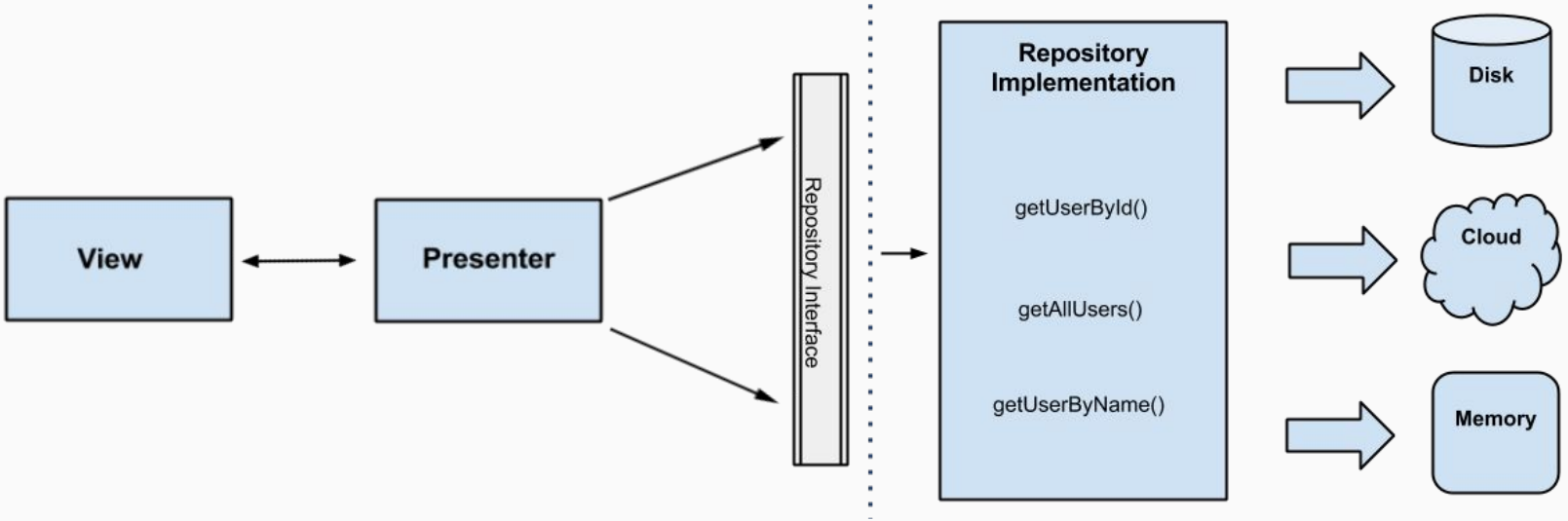
2) Controller может управлять несколькими View, а также переключать их, а Presenter управляет только одной View

Наша архитектура

1) Слой данных (Repository, кэширование)

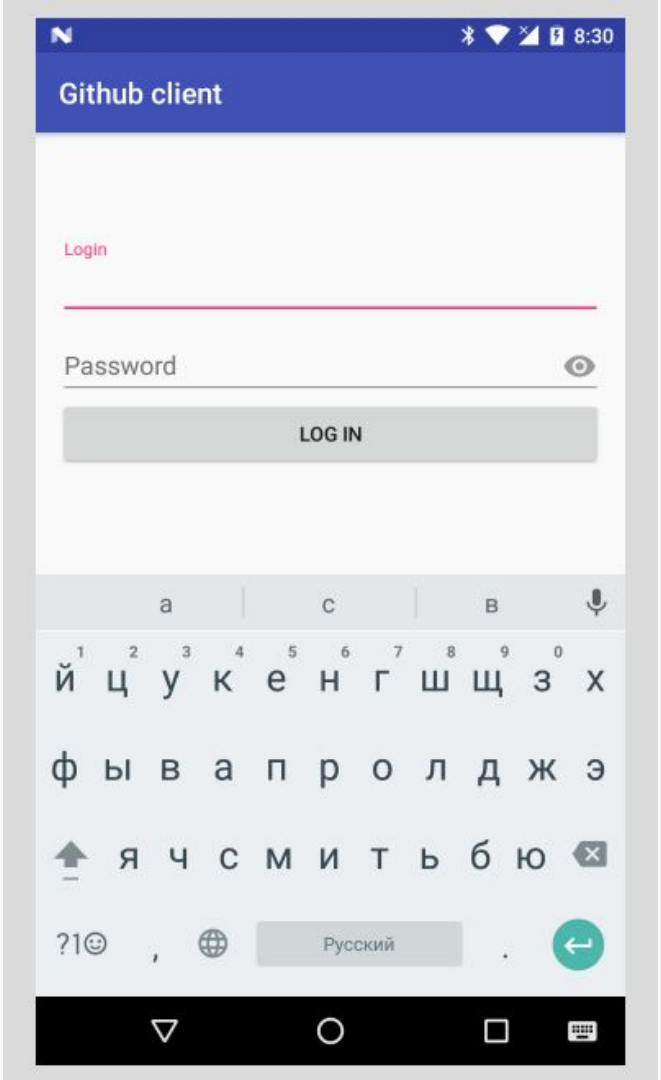
2) MVP

Наша архитектура



Пример экрана с MVP

Экран авторизации



Экран авторизации

1) Проверяем текущее состояние авторизации
2) Ошибка при нажатии кнопки, когда поля ввода пустые
3) Инициация процесса авторизации при выполнении запроса
4) Показ и скрытие процесса загрузки пользователю
5) Открытие главного экрана в случае успешной авторизации

AuthView

```
public interface AuthView extends LoadingView {

    void openRepositoriesScreen();

    void showLoginError();

    void showPasswordError();

}
```

LoadingView

```
public interface LoadingView {

    void showLoading();

    void hideLoading();

}
```

Реализуем интерфейс AuthView в AuthActivity

```java
@Override
public void showLoading() {
    mLoadingView.showLoading();
}

@Override
public void hideLoading() {
    mLoadingView.hideLoading();
}

@Override
public void showLoginError() {
    mLoginInputLayout.setError("Error");
}

@Override
public void showPasswordError() {
    mPasswordInputLayout.setError("Error");
}

@Override
public void openRepositoriesScreen() {
    RepositoriesActivity.start(this);
    finish();
}
```

AuthPresenter - поля

```
public class AuthPresenter {

    private final LifecycleHandler mLifecycleHandler;
    private final AuthView mAuthView;

    public AuthPresenter(@NonNull LifecycleHandler lifecycleHandler,
                         @NonNull AuthView authView) {
        mLifecycleHandler = lifecycleHandler;
        mAuthView = authView;
    }
```

AuthPresenter - запуск экрана

```
public void init() {
    String token = PreferenceUtils.getToken();
    if (!TextUtils.isEmpty(token)) {
        mAuthView.openRepositoriesScreen();
    }
}
```

AuthPresenter - обрабатываем нажатие кнопки входа

```
public void tryLogIn(@NonNull String login, @NonNull String password) {
    if (TextUtils.isEmpty(login)) {
        mAuthView.showLoginError();
    } else if (TextUtils.isEmpty(password)) {
        mAuthView.showPasswordError();
    } else {
        RepositoryProvider.provideGithubRepository()
                .auth(login, password)
                .doOnSubscribe(mAuthView::showLoading)
                .doOnTerminate(mAuthView::hideLoading)
                .compose(mLifecycleHandler.reload(R.id.auth_request))
                .subscribe(authorization -> mAuthView.openRepositoriesScreen(),
                        throwable -> mAuthView.showLoginError());
    }
}
```

Используем AuthPresenter

```
@OnClick(R.id.logInButton)
public void onLogInButtonClick() {
    String login = mLoginEdit.getText().toString();
    String password = mPasswordEdit.getText().toString();
    mPresenter.tryLogIn(login, password);
}
```

Вопросы

1) Насколько такая архитектура масштабируема?

2) Можно ли передавать Context в Presenter?

3) Нужно ли делать интерфейс или базовый класс для Presenter?

Практика

Практика

1) Проект GithubMVP

2) Нужно перевести экран walkthrough (описание в WalkthroughActivity) на MVP

3) Реализовать экран списка коммитов (описание в CommitsActivity) в соответствии с паттерном MVP и описанными сценариями

Дополнительно - библиотека Mosby

Зачем нужны библиотеки

1) MVP немного увеличивает код
2) Приходится писать много стандартного кода для каждого экрана

Будьте крайне осторожны!

1) Использование библиотеки для создания архитектуры нарушает первый из принципов Clean Architecture
2) Библиотека вынуждает вас писать код в ее рамках
3) Нужно хорошо изучить конкретное решение перед его использованием

Библиотеки

1) Mosby

2) Moxy

Преимущества Mosby

1) Структурирование кода и его организация в соответствии с паттерном MVP
2) Не нужно явно хранить View и Presenter в виде полей
3) Автоматическое связывание View и Presenter
4) LCE-экраны (Loading-Content-Error)

MvpView

```
public interface LoadingView extends MvpView {

    void showLoading();

    void hideLoading();

}
```

MvpBasePresenter

```
public class AuthPresenter extends MvpBasePresenter<AuthView> {

    private final LifecycleHandler mLifecycleHandler;

    public AuthPresenter(@NonNull LifecycleHandler lifecycleHandler) {
        mLifecycleHandler = lifecycleHandler;
    }
```

MvpBasePresenter

```
public void tryLogIn(@NonNull String login, @NonNull String password) {
    if (!isViewAttached() || getView() == null) {
        return;
    }

    if (TextUtils.isEmpty(login)) {
        getView().showLoginError();
    } else if (TextUtils.isEmpty(password)) {
```

MvpActivity

```
public class AuthActivity extends MvpActivity<AuthView, AuthPresenter> implements AuthView
```

```
@NonNull
@Override
public AuthPresenter createPresenter() {
    LifecycleHandler lifecycleHandler = LoaderLifecycleHandler.create(this, getSupportLoaderManager());
    return new AuthPresenter(lifecycleHandler);
}
```

```
@OnClick(R.id.logInButton)
public void onLogInButtonClick() {
    String login = mLoginEdit.getText().toString();
    String password = mPasswordEdit.getText().toString();
    getPresenter().tryLogIn(login, password);
}
```

Практика - 2

1) Проект GithubMosby
2) Нужно перевести экран walkthrough (описание в WalkthroughActivity) на MVP с использованием библиотеки Mosby
3) Реализовать экран списка коммитов (описание в CommitsActivity) в соответствии с паттерном MVP и с