

# INTRODUCTION TO MVVM

WITH KOTLIN AND ANDROID ARCHITECTURE COMPONENTS

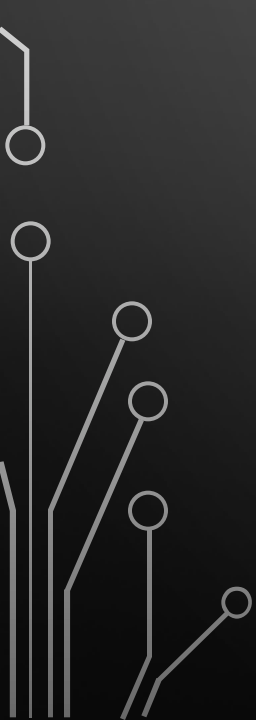
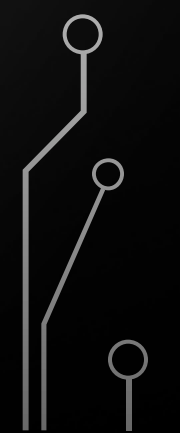


Abhishek Bansal  
Lead Android Developer  
DoctorBox Health



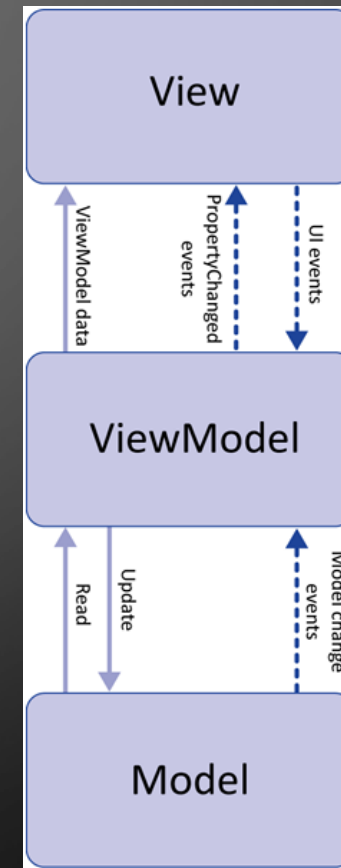
# AGENDA



- Introduction to MVVM
  - MVVM vs MVP
  - Architecture Components
    - View Model
    - LiveData
  - Demo
- 
- 

# INTRODUCTION TO MVVM

- Model-View-ViewModel
  - View receives user interaction and pass it on to ViewModel
  - ViewModel provides view with relevant data
  - ViewModel takes data from Model
- Model actually abstracts data source
- Data source is responsible for supplying data to ViewModel from Network or Cache

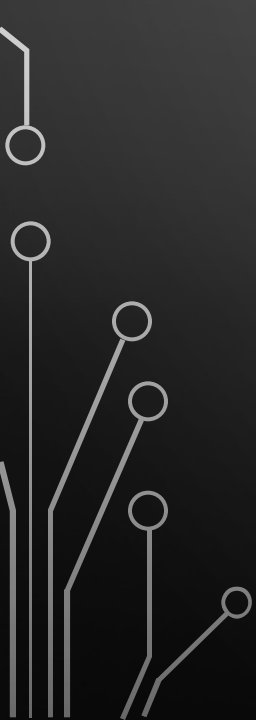
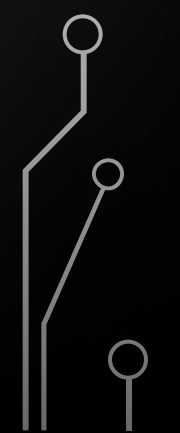


# MVVM VS MVP

- Both are similar in terms of abstraction of view and data
- MVVM pattern is more geared towards event driven programming
- In MVP Presenter tells View what to do
- In MVVM, ViewModel(Producer) exposes stream of data and View(Consumer) knows what to do with it.
  - This way ViewModel does not have to keep a reference of view.
  - Producer is not worried about how data is consumed



# ARCHITECTURE COMPONENTS- VIEWMODEL

- `android.arch.lifecycle.ViewModel`
    - Lifecycle Aware
    - Always created in association of scope i.e. Activity or Fragment
    - Doesn't get destroyed with configuration changes
    - Can be shared between Fragments and Activities
- 
- 

# ARCHITECTURE COMPONENTS- VIEWMODEL

```
public class UserActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity_layout);
        final UserModel viewModel = ViewModelProviders.of(this).get(UserModel.class);
        viewModel.userLiveData.observe(this, new Observer() {
            @Override
            public void onChanged(@Nullable User data) {
                // update ui.
            }
        });
        findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                viewModel.doAction();
            }
        });
    }
}
```

```
public class UserModel extends ViewModel {
    public final LiveData<User> userLiveData = new LiveData<>();

    public UserModel() {
        // trigger user load.
    }

    void doAction() {
        // depending on the action, do necessary business logic calls and update the
        // userLiveData.
    }
}
```

Source:

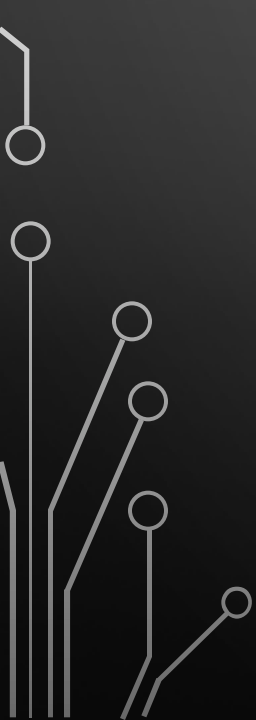
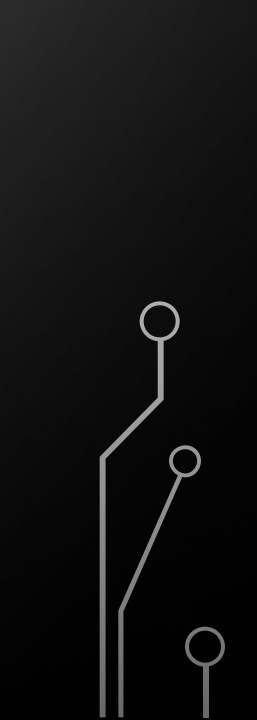
<https://developer.android.com/reference/android/arch/lifecycle/ViewModel>

# ARCHITECTURE COMPONENTS- LIVEDATA

- `android.arch.lifecycle.LiveData<T>`
  - Data Holder class with a twist
  - Its an Observable
  - Lifecycle aware, Clean up themselves automatically



# ARCHITECTURE COMPONENTS- LIVEDATA

- Benefits over RxJava Observables
    - Ensures your UI matches your data state
    - No memory leaks
    - No crashes due to stopped activities
    - No more manual lifecycle handling
    - Always up to date data
    - Proper configuration changes
    - Sharing resources
- 
- 





DEMO

# FURTHER READINGS/REFERENCES

- MVVM

- <https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel-e7eeee76b73b>
- [http://upday.github.io/blog/mvvm\\_rx\\_common\\_mistakes/](http://upday.github.io/blog/mvvm_rx_common_mistakes/)

- ViewModel

- <https://developer.android.com/reference/android/arch/lifecycle/ViewModel>

- LiveData

- <https://developer.android.com/reference/android/arch/lifecycle/LiveData>