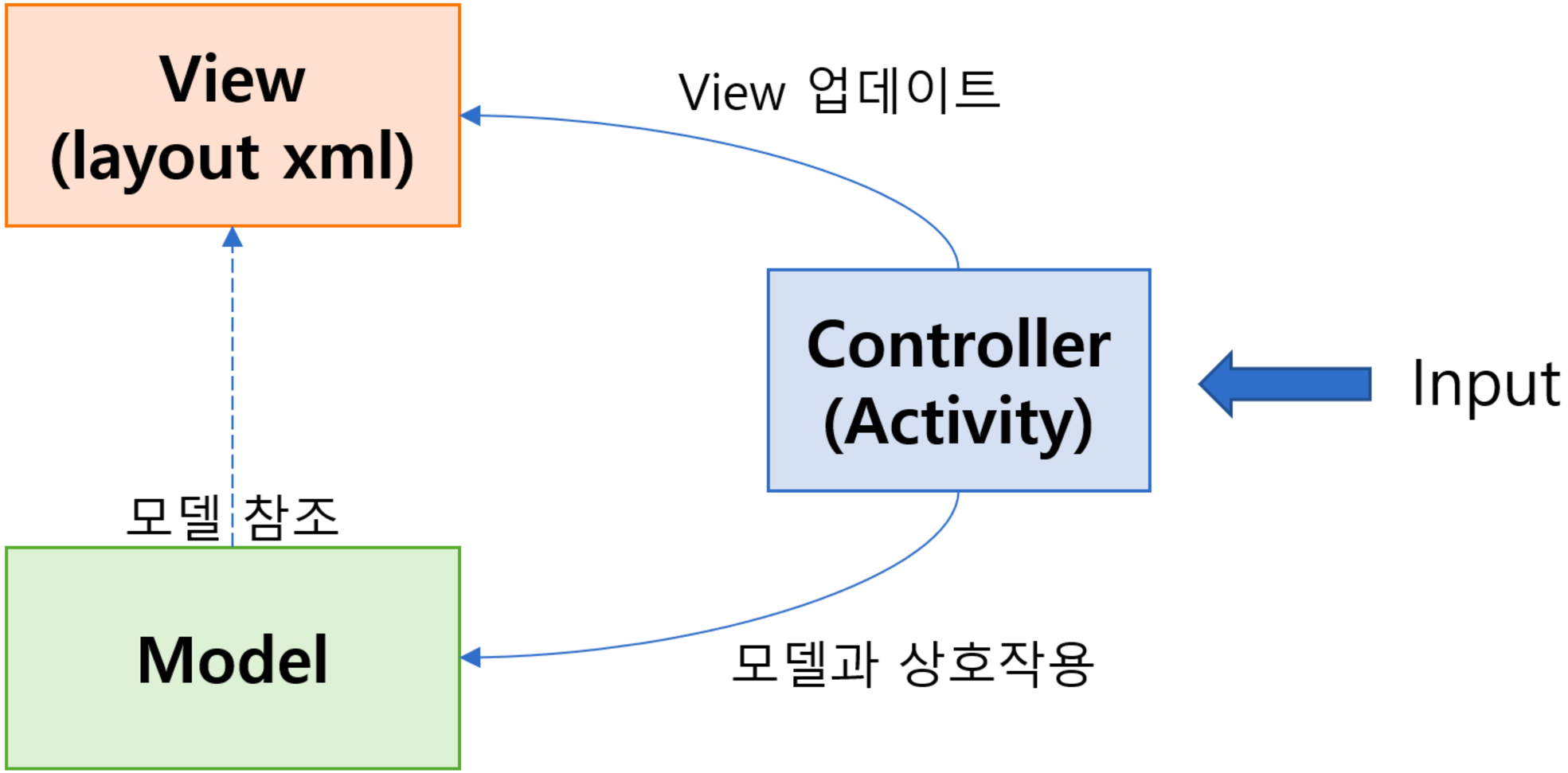
플러터 다트 아키텍처 패턴

1. MVC 패턴

MVC 패턴이란?

- 코드를 각각 Model-View-Controller로 나누어 모델과 뷰를 분리하여 종속성을 낮추고 확장성과 재사용성을 높이면서 유지보수와 단위 테스트가 간편해지도록 구성하는 아키텍처 디자인 패턴
- Model은 데이터와 비즈니스 로직을 담당하며
- View는 사용자에게 보여지는 UI
- Controller는 Model과 View의 메신저 같은 역할을 함

참조: 깃뚜님의 블로그
https://gitsu.tistory.com/38



참조: 깃뚜님의 블로그
https://gitsu.tistory.com/38

MVC 패턴의 단점

- 컨트롤러의 역할이 많기 때문에 앱이 커질수록 컨트롤러 비대화 발생
- View와 Model이 완전 독립적이지 않음

참조: 깃뚜님의 블로그
https://gitsu.tistory.com/38

플러터와 MVC 패턴

- 웹 개발이나 네이티브 개발과 달리 플러터는 UI와 코드를 완전 분리할 수가 없음
- 이유는 플러터 앱은 위젯의 조합으로 이루어지고 각각의 위젯이 View에 해당하는 UI 부분과 Controller에 해당하는 이벤트 리스너를 동시에 들고 있기 때문
- 예를 들어 IconButton은 UI에 해당하는 size, color 등의 속성과 controller에 해당하는 onPressed 콜백을 동시에 들고 있음
- 물론 리액트의 컴포넌트나 안드로이드의 액티비티, 프래그먼트 또한 이러한 성질을 지니고 있으나 다음 장에 설명할 부분에서 결정적인 차이를 보임

안드로이드 예시 - xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="50dp">

    <TextView
        android:id="@+id/name"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:gravity="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/delete"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/delete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="DELETE"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

참조: 팀노바 & Stickode 개발자 블로그
https://stickode.tistory.com/245

안드로이드 예시 - java

```java
public class MainActivity extends AppCompatActivity {

    private Model model;
    private TextView textView;

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

        //View 작성
        setContentView(R.layout.activity_main);
        textView = findViewById(R.id.text_view);

        //Controller 작성
        model = new Model();
        textView.setOnClickListener(view -> {
            textView.setText(model.clikedButton());//Model 부
분

        });
    }
}
```

참조: 깃뚜님의 블로그
https://gitsu.tistory.com/38

안드로이드 예시

- 하지만 플러터는 다른 파일에서 생성한 위젯을 다른 파일에서 접근할 수도 없고 굳이 할 필요도 없음. 따라서 해당 위젯에서 코드를 해결해야 함
- 물론 Controller를 View 코드와 함께 사용할 수 있음. 그러나 그렇게 하면 View는 Controller를 몰라야한다는 MVC의 원칙에 위배됨
- 반대로 Model을 View에 주입해서 사용할 수 있음. 하지만 그렇게 되면 View는 Model을 알 수 있으나 Controller를 통해 주입 받아야한다는 원칙을 어기게 됨
- 여기서 View는 Controller를 몰라야한다는 첫 번째 문장과 View는 Controller를 통해 Model을 주입 받아야한다는 것이 모순처럼 들릴 수 있으나 Controller는 View를 알 수 있기 때문에 Controller 코드에서 View에 Model을 주입하는 것이기 때문에 모순되지 않음

코드 시청 시간

결론

- 플러터의 경우 MVC 패턴을 엄격히 적용하기가 사실상 불가능
- 이런 이유로 현재 플러터+MVC를 이야기하는 컨텐츠가 별로 없고 있어도 제대로 설명하는 것을 찾기 힘듦
- 그러나 MVC의 경우 웹에서 발전되어 왔고 아직도 웹에서 사랑 받는 패턴이기 때문에 플러터에 굳이 적용할 필요가 없음
- 이미 네이티브 안드로이드, iOS 또한 컴포즈 및 스위프트UI로 변화하는 과도기이기 때문에 차세대 개발 방식의 플러터에 더 적합한 패턴을 찾아 개발하는 것이 더욱 중요