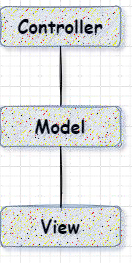
**MVC** is known as an architectural pattern, which embodies three parts Model, View and Controller, or to be more exact it divides the application into three logical parts: the model part, the view and the controller. It was used for desktop graphical user interfaces but nowadays is used in designing mobile apps and web apps.

## History:

MVC was created by Trygve Reenskaug. When he was a visiting scientist at Xerox Palo Alto Research Laboratory (PARC) in 1978/79, he wrote the initial reports on MVC. MVC was originally known as "Thing Model View Editor," but it was soon changed to "Model View Controller."

Tygrve was designed to address the issue of users controlling a large and complex data set. MVC as a technique has evolved over the years. The MVC pattern was first used as an architectural pattern for graphical user interfaces because it was created before web browsers (GUI).



The Original MVC

**MVC Architecture**

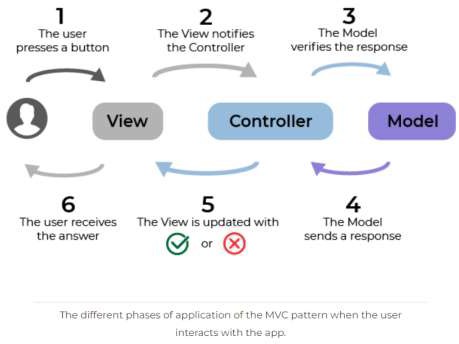
MVC is an architectural pattern, which indicates it controls the application's entire architecture. Even though it is commonly referred to as a design pattern, we may be wrong if we refer to it entirely as such. Design patterns are used to solve a specific technical problem, whereas architecture patterns are used to solve architectural problems, and thus affect the entire architecture of our application.

It is made up of three primary parts:

* Model
* View
* Controller

each with their own set of responsibilities. MVC is used for two primary reasons:

* It stops us from repeating ourselves.
* It contributes to the growth of a strong web application framework.



*Image taken from* [*this*](https://openclassrooms.com/en/courses/4661936-develop-your-first-android-application/4679186-learn-the-model-view-controller-pattern) *article which helps to understand how MVC works.*

# Model:

It is considered as a low level because it is responsible for data management. Handle data logically, therefore it's all about data. Because the model is linked to the database, anything you do with data will be represented in the model. The model component is where you add or retrieve data. Because the controller never connects to the database by itself, it reacts to controller requests. The model interfaces with the database and then provides the required data to the controller. It's important to note that the model never directly communicated with the view.

## View:

The view component is in control of data representation. It creates the user interface, or UI, for the user. So, when you think of a view component in a web application, only think of the HTML/CSS section. The data acquired by the model component is used to generate views, but these data are passed through the controller rather than directly to the view, so the view only communicates with the controller.

## Controller:

Because the controller is the component that allows the interconnection between the views and the model, it is known as the "main man." The controller doesn't have to handle data logic; all it has to do is inform the model what to do. After getting data from the model, it processes it before sending it all to the display and explaining how to represent it to the user. Views and models cannot interact directly.

# KeyPoints:

So MVC isn't easy to understand; in fact, it's really difficult to understand, but it's not impossible to learn, and every developer should keep it in mind when designing an application. Keep in mind that MVC is a software architecture that breaks down your software into smaller pieces.

* The model is concerned with the facts and logic of your system.
* The controller maintains the link between the model and
* the view, while the view only displays data. This 'division' improves the user experience and modularity while also making testing easier.

# Advantages of MVC:

* The components can be utilised.
* Easy to maintain too.
* In MVC, different components of the application can be published and maintained individually.
* This architecture helped in the independent testing of components.

# Disadvantages of MVC:

* The level of difficulty is high.
* This product is not suitable for tiny applications.
* In view is the inefficiency of data access.

# MVC EXAMPLE:

Let’s try to understand the MVC architecture pattern using a very simple android application. Here we will be building an app for login or authentication. We will be using Kotlin but you can also use Java.

### Step 1. - Create a new Android Project

Click on your Android Studio icon and create a new project with an empty activity. You can name your project anything you like, we named it as MVC Example.

### Step 2. – Create three packages

To implement MVC architecture in this project we have to create three different packages-

1. **Model**
2. **View**
3. **Controller**

Each package will be containing classes and interfaces. To create these package folders –

Right Click on the project name > New > Package

### Step 3. – Create interfaces and classes

Now, we have to create files and interfaces in the package we created above. In Model Package create two files-

**iUser.kt**

package com.prabhjot.mvcexample.Model

interface iUser {

fun getEmail(): String? fun getPassword(): String? fun isValid(): Int

}

**User.kt**

package com.prabhjot.mvcexample.Model

import android.text.TextUtils import android.util.Patterns

class User(

private val email: String?, private val password: String?

) : iUser {

override fun getEmail(): String? { return email

}

override fun getPassword(): String? { return password

}

override fun isValid(): Int {

if (TextUtils.isEmpty(getEmail())) return 0

else if (!Patterns.*EMAIL\_ADDRESS*.matcher(getEmail()).matches()) return 1

else if (TextUtils.isEmpty(getPassword())) return 2

else if (getPassword()?.length!! <= 6) return 3

else

return -1;

}

}

Now, for the controller package, we have to create one interface iLogin, and a class named as Login.

### iLogin.kt

package com.prabhjot.mvcexample.Controller

interface iLogin {

fun OnLogin(email: String?, Password: String?)

}

**Login.kt**

package com.prabhjot.mvcexample.Controller

import com.prabhjot.mvcexample.Model.User import com.prabhjot.mvcexample.View.LoginView

class Login(

private val loginView: LoginView

) : iLogin {

override fun OnLogin(email: String?, password: String?) { val user = User(email, password)

val loginCode = user.isValid()

when (loginCode) { 0 -> {

loginView.OnLoginError("Enter Email");

}

1 -> {

loginView.OnLoginError("Enter a valid Email");

}

2 -> {

loginView.OnLoginError("Enter Password");

}

3 -> {

loginView.OnLoginError("Enter Password greater the 6 char");

}

else -> {

loginView.OnLoginSuccess("You logged in.");

}

}

}

}

We are done with model and controller. Now, its time to design UI using View so that data will be displayed in the views.

**LoginView.kt**

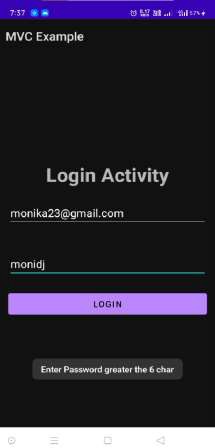
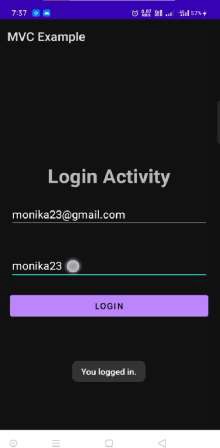
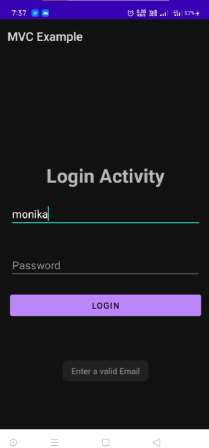
package com.prabhjot.mvcexample.View

interface LoginView {

fun OnLoginSuccess(message: String?) fun OnLoginError(message: String?)

}.

### Output –



*Screen shots of the output or the debugged app which we got from the above code.*

With this example, we learned how we can implement MVC architecture in an Android App using Kotlin. Below are some recommended posts by which you can understand MVC examples more clearly.

**More MVC examples or recommended Posts** [MVC Architecture Pattern in Android with Example.](https://www.geeksforgeeks.org/mvc-model-view-controller-architecture-pattern-in-android-with-example/) [Use MVC pattern to create very basic Shopping Cart](https://androidexample.com/Use_MVC_Pattern_To_Create_Very_Basic_Shopping_Cart__-_Android_Example/index.php?view=article_discription&aid=116) [Implementing MVC pattern in Android with Kotlin](https://www.codementor.io/%40dragneelfps/implementing-mvc-pattern-in-android-with-kotlin-i9hi2r06c)

[The Model View Controller – MVC Architecture and Frameworks Explained.](https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/)

# Conclusion:

So MVC isn't easy to understand; in fact, it's really difficult to understand, but it's not impossible to learn, and every developer should keep it in mind when designing an application. Keep in mind that MVC is a software architecture that breaks down your software into smaller pieces. The model is concerned with the facts and logic of your system. The controller maintains the link between the model and the view, while the view only displays data. This 'division' improves the user experience and modularity while also making testing easier.

# Resources (articles):

[Wikipedia - MVC](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)

[Notes and Historical documents](http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html) from Trygve Reenskaug, inventor of MVC.

[The Evolution of MVC and other UI architectures from Martin Fowler](http://martinfowler.com/eaaDev/uiArchs.html).

[Model View Controller Theory](https://www.raywenderlich.com/books/advanced-android-app-architecture/v1.0/chapters/2-model-view-controller-theory)

[What is wrong with MVC](https://cocoacasts.com/what-is-wrong-with-model-view-controller) [Benefits of using MVC](https://www.geeksforgeeks.org/benefit-of-using-mvc/)

[What is MVC, and how is it like a sandwich shop?](https://www.freecodecamp.org/news/simplified-explanation-to-mvc-5d307796df30/)

**Resources (videos/ tutorials):**

[The Principles of Clean Architecture by Uncle Bob Martin](https://youtu.be/o_TH-Y78tt4)

[MVC Explained in 4 minutes.](https://www.youtube.com/watch?v=DUg2SWWK18I)

[Why ASP.NET MVC and MVC vs WebForms ?](https://www.youtube.com/watch?v=bGpBgDDDVlM)