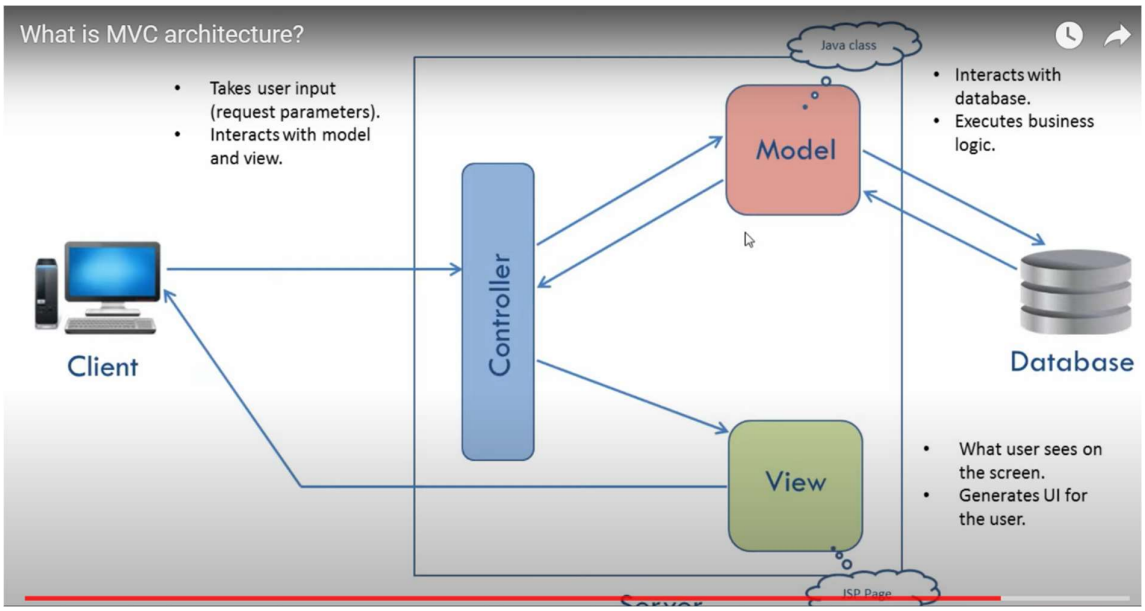
# MVC nedir?

MVC *(Model-View-Controller)*, yazdığımız uygulamanın iş mantığı ile (business logic) kullanıcı arayüzünü birbirinden ayrıştıran, uygulamanın farklı amaçlara hizmet eden kısımlarının birbirine girmesini engelleyen yazılım mimarisidir.

Kodun farklı amaçlara hizmet eden yapılarını birbirinden ayırarak, kodu daha rahat geliştirilebilir ve test edilebilir (yani daha az hata çıkartma potansiyeline sahip) hale getirmiş oluruz.




**Model**

- Uygulamada kullanılan verileri temsil eder ve verilerin işlenme **mantığının** saklandığı kısımdır. (Verilerin validasyonu burada yapılır)
- *Genelde* verilerin veritabanı (veya XML gibi benzer bir yere) kaydedilmesi ve kayıtlı yerden alınması işlemleri yine burada olabilir.

**View**

- Basitçe, uygulamanızın kullanıcılarınızın gözüyle gördüğü kısmıdır, arayüzdür.

**Controller**

- Model ve View arasında getir götür işlemlerini gerçekleştirir.
- Kullanıcıların View üzerinden gerçekleştirdiği işlemlerle alınan veriyi Model'e taşır, Model'den aldığı veriyi View üzerinden kullanıcıya gösterir.
- MVC yapısında ana mantık Model ve View yapısının ayrılmasıdır. Bu iki yapı arasındaki haberleşmeyi sağlayan köprüye Controller diyoruz.

Peki neden MVC mimarisini kullanmalıyız?

MVC sayesinde Model ve View yapısını ayrıştırmış oluyoruz. Böylelikle yarın bir gün uygulamamızın görünümünü değiştirmek durumunda kaldığımızda "yalnızca" görünümle uğraşmamız gerekecek. İç içe geçmiş, spagetti bir kodla uğraşmak durumunda kalmış olsaydık, sadece görünümü değiştirmek isterken uygulamanın "işleyişini" de değiştirmemiz gerekecekti. Ayrıca, bu ayrıştırma sayesinde Model ve View yapımızda ihtiyaç duyduğumuz parçaları, başka projelerde de tekrar kullanılabilir hale getirmiş olduk. Sonuçta, yazılım geliştirmede yegane amacımız "hatadan uzak olmak" ve "zamandan tasarruf etmek".

MVC nedir sorusunu cevaplayan, Programmers – StackExchange'de karşılaştığım şöyle güzel bir örnekten alıntı yapabilirim:

*Satranç uygulaması yaptığımızı düşünelim…*

*Model'de oyunun "durumu" barındırılacaktır. Oyunun durumunu değiştirecek etkiler (örneğin bir taşın hareketinin doğru olup olmadığı) veya oyunun bitip bitmediği gibi bilgiler model*

*üzerinde yer alacaktır. (Kısaca, oyunun tüm bilgileri ve yapılacak işlemlerin validasyonu Model üzerinde barınacak)*

*View kısmında satranç tahtasının görünümü, yönettiğimiz piyonların şekilleri ve piyonları hareket ettirdiğimizde hangi piyonun nereye gittiğini söyleyen bildirimler yer alacaktır. Piyonların nasıl hareket ettiği, oyunun durumuyla ilgili mantıksal bilgilerin View ile hiçbir işi olmayacaktır. View sadece ve sadece görselliği barındıracaktır.*

*Controller ise View ve Model arasında haberleşmeyi sağlayacaktır. Örneğin, kullanıcı View'da yer alan "Yeni oyun başlat" butonuna bastığında Controller, Model'e giderek böyle bir isteğin geldiğini söyleyecektir. (Tüm bu işleri **yapan** Model olacaktır, Controller'ın amacı böyle bir isteğin geldiğini ve alakalı isteğin detaylarını Model'e **iletmektir**)*

*Peki kazancımız ne oldu?*

- *Model veya View üzerinde değişiklik yapmak istediğimizde bu değişiklikler Model veya View'u artık etkilemeyecek. Yapılan bu değişiklikler Controller'ın yapısını değiştirmemize sebep verebilir ama hiç değilse değişiklik yapacağımız yeri sabit kısımlardan ayırmış olup spagetti kodu engellemiş olduk.*
- *Model ya da View'u tekrar kullanabilir hale getirdik. Örneğin Model olarak RSS feed'i kullanıp View'u sabit tutarak, daha önceden oynanmış oyunları hazırladığımız View üzerinde gösterebiliriz. (Replay misali) Veya View'u değiştirip Model'i sabit tutarak oyunu hem Web sitesi üzerinden hem de Konsol uygulaması üzerinden oynanabilir hale getirebiliriz.*
- *Hem View hem de Model'i iyi ayrıştırdığımız için bu yapılara Ünite testi yazmayı da kolaylaştırmış olduk.*

Controller bizim için yalnızca aracı görevi görüyor. İş mantığı Model'de, görsel mantık View'da olmalı, Controller sadece haberleşmeyi sağlamalı. Controller'a, Model'in ve/ya View'ın sorumlulukları yüklenirse MVC kullanmanın hiçbir anlamı yok.

MVC tasarım desenine dayalı bir web uygulaması uygulamak için:

- Model katmanı görevi gören Kurs Sınıfı
- Sunum katmanını tanımlayan CourseView Sınıfı (görünüm katmanı)
- Denetleyici görevi gören CourseContoller Sınıfı

Şimdi bu katmanları tek tek inceleyelim.

## Model Katmanı

MVC tasarım modelinde model, sistemin iş mantığını tanımlayan ve aynı zamanda uygulamanın durumunu temsil eden veri katmanıdır. Model nesneleri, modelin durumunu bir veritabanında alır ve saklar. Bu katman aracılığıyla, sonunda uygulamamızın yönettiği kavramları temsil eden verilere kurallar uygularız. Şimdi Course Class kullanarak bir model oluşturalım.

```
3  package MyPackage;
4
5  public class Course {
6      private String CourseName;
7      private String CourseId;
8      private String CourseCategory;
9
10     public String getId() {
11       return CourseId;
```

```
12      }
13
14      public void setId(String id) {
15        this.CourseId = id;
16      }
17
18      public String getName() {
19        return CourseName;
20      }
21
22      public void setName(String name) {
23        this.CourseName = name;
24      }
25
26      public String getCategory() {
27            return CourseCategory;
28        }
29
30      public void setCategory(String category) {
31            this.CourseCategory = category;
          }
    }
```

## View Katmanı

Uygulamanın veya kullanıcı arayüzünün çıktısını temsil eder. Kontrolör tarafından model katmanından alınan verileri görüntüler ve istendiğinde kullanıcıya sunar. İhtiyaç duyduğu tüm

bilgileri denetleyiciden alır ve iş katmanıyla doğrudan etkileşime girmesi gerekmez. CourseView Class kullanarak bir view oluşturalım.

```
ackage MyPackage;


public class CourseView {
    public void printCourseDetails(String CourseName, String CourseId, String CourseCategory){
        System.out.println("Course Details: ");
        System.out.println("Name: " + CourseName);
        System.out.println("Course ID: " + CourseId);
        System.out.println("Course Category: " + CourseCategory);
    }
  }
```

## Controller Katmanı

Denetleyici, Model ve Görünüm arasındaki bir arayüz gibidir. Görünüm katmanından kullanıcı isteklerini alır ve gerekli doğrulamaları da içerecek şekilde işler. İstekler daha sonra veri işleme için modele gönderilir. İşlendikten sonra veriler tekrar kontrolöre geri gönderilir ve ardından görünümde görüntülenir. Denetleyici görevi gören CourseContoller Sınıfı oluşturalım.

```
1 package MyPackage;
2
3 public class CourseController {
4     private Course model;
5     private CourseView view;
6
7     public CourseController(Course model, CourseView view){
8       this.model = model;
```

```
9         this.view = view;
10    }
11
12    public void setCourseName(String name){
13        model.setName(name);
14    }
15
16    public String getCourseName(){
17        return model.getName();
18    }
19
20    public void setCourseId(String id){
21        model.setId(id);
22    }
23
24    public String getCourseId(){
25        return model.getId();
26    }
27
28    public void setCourseCategory(String category){
29            model.setCategory(category);
30    }
31
32        public String getCourseCategory(){
33            return model.getCategory();
34    }
35    public void updateView(){
```

```
36          view.printCourseDetails(model.getName(), model.getId(), model.getCategory());
        }
    }
```

## KAYNAKLAR:

1-MVC Architecture in Java: How to implement MVC in Java? Edureka