



A Beginner's Guide to MVC Architecture in Java



by **Arjun Mathur**

[Home](#) > [Software Development](#) > [Technology](#) > A Beginner's Guide to MVC Architecture in Java

APR 9, 2018

If you're related to web development scene, you'll most certainly have come across the acronym "MVC" hundreds of time, if not more. MVC is one of the most talked about design patterns in the web programming world today, and rightly so. In this article, we'll take you through the world of MVC Architecture. We'll talk about the basics of MVC, its advantages, and finally, a simple code that will help you understand the implementation of [MVC](#) in Java in a clearer way.

What exactly is the MVC architecture?

Before we get into the technicalities of it, let's make some things clear – MVC is NOT a design pattern, it's a way to structure your applications. In recent years, the web applications are extensively using the MVC architecture, and hence it's natural to confuse it for a design pattern exclusive for web applications. However, let's tell you that MVC was first described way back in 1979 – before the WWW era when there was no concept of web applications. The architecture used today for web applications is an adaptation of the original pattern.

In fact, this architecture was initially included in the two major web development frameworks – Struts and Ruby on Rails. These two environments paved the way for a majority of web frameworks that came later – and thus the popularity of this architecture kept rising.

Why Companies are Looking to Hire Full Stack Developers [↗](#)

MVC architectural pattern follows an elementary idea – we must separate the responsibilities in any application on the following basis:

Most Popular

[IntelliJ IDEA v](#)
[The Holy War!](#)

[What Does A S](#)
[Developer Do?](#)

[A Beginner's G](#)
[MVC Architect](#)

Editor's Picks

[Best Career Op](#)
[Graduation – I](#)
[2019](#)

[How to becom](#)
[Learning Engi](#)
[Steps \(with pic](#)

[6 Interesting M](#)
[Learning Proje](#)
[For Beginners](#)



- **Controller:** Entertains user requests and fetch necessary resources.

Each of the components has a demarcated set of tasks which ensures smooth functioning of the entire application along with complete modularity. Let us have a look at each of these components in further detail.

Controller

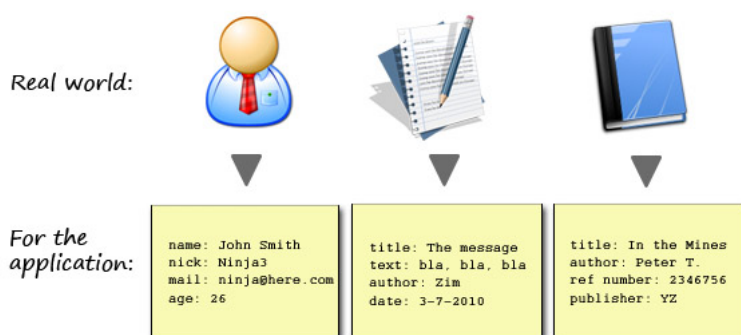
The controller is the like the housekeeper of the application – it performs coordination between model and view to entertain a user request. The user requests are received as HTTP get or post request – for example, when the user clicks on any GUI elements to perform any action.

The primary function of a controller is to call and coordinate with the model to fetch any necessary resources required to act. Usually, on receiving a user request, the controller calls the appropriate model for the task at hand.

Model

The model is quite simply the data for our application. The data is “modelled” in a way it’s easy to store, retrieve, and edit. The model is how we apply rules to our data, which eventually represents the concepts our application manages.

For any software application, everything is modelled as data that can be handled easily. What is a user, a book, or a message for an app? Nothing really, only data that must be processed according to specific rules. Like, the date must not be higher than the current date, the email must be in the correct format, the name mustn't be more than “x” characters long, etc.



Whenever a user makes any request from the controller, it contacts the appropriate model which returns a data representation of whatever the user requested. This model will be the same for a particular work, irrespective of how we wish to display it to the user. That is why we can choose any available view to render the model data.

Additionally, a model also contains the logic to update the relevant controller whenever there is any change in the model's data.

View

As the name suggests, the view is responsible for rendering the data received from the model. There may be pre-designed templates where you can fit the data, and there may even be several different views per model depending on the requirements.

With UpGrad

By clicking 'Submit' you agree to our [UpGrad's Terms & Conditions](#)

Any web application is structured keeping these three core components in mind. There may be a primary controller that is responsible for receiving all the requests and calling the specific controller for specific actions. Let's understand the working of a web application under the MVC architecture better using an example.

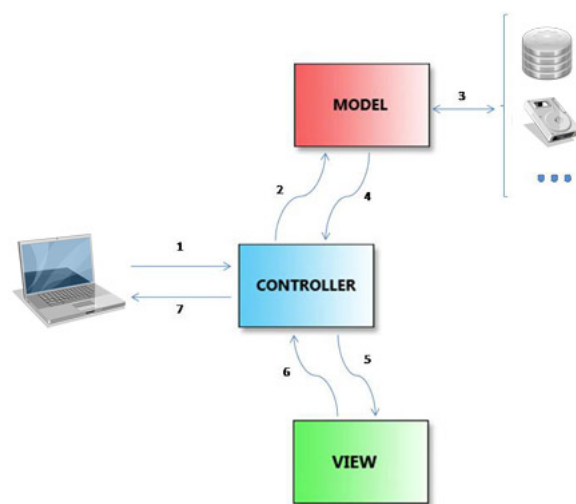
Let's See an Example

Let's take an example of an online stationery shop. The user can view items, buy, add items to cart, add items to current order, or even add/remove items (if he's the admin).

Now, let's see what will happen when a user clicks on the title "Pens" to see the list of pens.

Our application will have a particular controller to handle all the queries related to pens. Let's say it's called "pens_controller.php". We'll also have a model that will store the data regarding the pens we have – let's call it "pens_model.php". Finally, we'll have several views to present the data – a list of pens, a table displaying pens, a page to edit the list, etc.

The following figure shows the complete [flow of control](#) right from the moment a user clicks on "pens", to when the result is rendered in front of him:



First, the "pens_controller.php" handles the user request (1) as a GET or POST request. We can also have an "index.php" which is the central controller which will call the "pens_controller" whenever needed.

The controller then examines the request and the parameters and calls the required model – in this case, "pens_model.php". The controller asks the model to return the list of available pens (2).

Now, the model searches the database for the necessary information (3), applies logics if necessary, and returns the data to the controller(4).

The controller then picks an appropriate view (5) and presents the data (6 and 7). If a request comes from a handheld device, a view suitable for it will be used, or if the user has a particular theme selected, its view will be picked – and so on.

The Advantages of the MVC Architecture

A common problem faced by application developers these days is the support for different type of devices. The MVC architecture solves this problem as developers can create different interfaces for different devices, and based on from which device the request is made, the controller will select an appropriate view. The model sends the same data irrespective of the device being used, which ensures a complete consistency across all devices.

The MVC separation beautifully isolates the view from the business logic. It also reduces complexities in designing large application by keeping the code and workflow structured. This makes the overall code much easier to maintain, test, debug, and reuse.

A Simple Implementation of MVC using Java

We will have the following three:

1. StudentObject : the model.
1. StudentView: view class to print details on the console.
1. StudentController: a controller that stores data in studentObject and updates StudentView accordingly

Step 1: Create the Model



```
public String getRollNo() {  
    return rollNo;  
}  
  
public void setRollNo(String rollNo) {  
    this.rollNo = rollNo;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
}
```

The code is self-explanatory. It consists of functions to get/set roll number and names of the students.

Let's call it "student.java".

Step 2: Create the View

```
public class StudentView {  
    public void printStudentDetails(String studentName, String studentRollNo){  
        System.out.println("Student: ");  
        System.out.println("Name: " + studentName);  
        System.out.println("Roll No: " + studentRollNo);  
    }  
}
```

This is simply to print the values to the console. Let's call this "studentView.java".

Step 3: Create the Controller



```

public StudentController(Student model, StudentView view){
    this.model = model;
    this.view = view;
}

public void setStudentName(String name){
    model.setName(name);
}

public String getStudentName(){
    return model.getName();
}

public void setStudentRollNo(String rollNo){
    model.setRollNo(rollNo);
}

public String getStudentRollNo(){
    return model.getRollNo();
}

public void updateView(){
    view.printStudentDetails(model.getName(), model.getRollNo());
}
}

```

Call this “StudentController.java”. A cursory glance will tell you that this controller is just responsible for calling the model to get/set the data, and then updating the view.

Now, let's have a look at how all of this is tied together.

Step 4: Create the main Java file

```

public class MVCPatternDemo {
    public static void main(String[] args) {
        //fetch student record based on his roll no from the database
        Student model = retrieveStudentFromDatabase();
        //Create a view : to write student details on console
        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);
        controller.updateView();
        //update model data
        controller.setStudentName("John");
        controller.updateView();
    }

    private static Student retrieveStudentFromDatabase(){
        Student student = new Student();
        student.setName("Robert");
        student.setRollNo("10");
        return student;
    }
}

```

This is called “MVCPatternDemo.java”. As you can see, it fetches the student data from the database or a function (in this case we're using a function to set the values) and pushes it on to the Student model. Then, it initialises the view we had created earlier.

Further, it also initialises our controller and binds it to the model and the view. The updateView() method is a part of the controller which updates the student details on the console.



Student:

Name: Robert

Roll No: 10

Student:

Name: John

Roll No: 10

If you get this as the output, congratulations! You've successfully implemented the MVC architecture using Java, albeit for a simple application. However simple, this application is enough to demonstrate the powers of the MVC architecture.

15 Must-Know Spring MVC Interview Questions [↗](#)

Wrapping Up...

After so much said and done, it's hard to emphasise more on the power of the MVC architecture in any web/desktop application these days. It provides an altogether new level of modularity to your code which makes it a lot more readable and maintainable. So, if you want to scale to new heights in your career as an application developer, getting hold of the MVC architecture and its working should be your top priority.

Do drop by a comment and let us know how you like this article!

About

Latest Posts



Arjun Mathur

Arjun is Program marketing manager at UpGrad for the Software development program. Prior to UpGrad, he was a part of the French ride-sharing unicorn "BlaBlaCar" in India. He is a B.Tech in Computers Science from IIT Delhi and loves writing about technology.

[SOFTWARE](#)

[TECHNICAL SKILLS](#)

[WEB DEVELOPMENT](#)


Become a Full Stack Developer

UPGRAD AND IIIT-BANGALORE'S PG DIPLOMA IN SOFTWARE DEVELOPMENT

[LEARN MORE](#)

Related Articles






Java Interview
Questions & Answers

[Top 21 Java Interview Questions & Answers for Freshers 2019](#)


by upGrad Jun 24, 2019



Most In-demand Jobs
for Software Engineers

[8 In Demand Careers Options for Software Engineers](#)

by upGrad Jun 19, 2019



15 Must Know Agile Methodology
Interview Questions

by Arjun Mathur Jun 10, 2019

Building Careers of Tomorrow

DATA SCIENCE

Data Science | PG Program

Machine Learning and AI | PG Program

Big Data | PG Program

Data Science | PG Certificate

Machine Learning and NLP | PG Certificate

Machine Learning and Deep Learning | PG Certificate

MANAGEMENT

Digital Marketing | PG Certificate

Product Management | PG Certificate

Strategic Digital Marketing | Executive Program

Entrepreneurship | PG Certificate

Management | PG Program

Insurance | PG Program

TECHNOLOGY

Software Development – Blockchain | PG Program

Product Management | PG Certificate

Software Development | PG Program

Blockchain Technology | PG Certificate

Big Data | PG Program

Entrepreneurship | PG Certificate

CATEGORIES

- > Data
 - > Artificial Intelligence
 - > Big Data
 - > Data Science
 - > Machine Learning
- > General
- > Management
 - > Digital Marketing
 - > Entrepreneurship
 - > Marketing Strategies
- > Product Management
- > Technology
 - > Blockchain Technology
 - > Software Development

RECENT POSTS

[Learn Data Science – AI become Data Scientist](#)

[8 Product Manager Interview Answers – Frequently Asked](#)

[Data Analyst Salary in India Seniors – Unbelievable](#)

[5 Trending Professional Graduation – Job Orientation](#)

[The Role of Digital Marketing Campaigns](#)

UPGRAD

[Contact Us](#)

[Privacy Policy](#)

[Terms & Conditions](#)

© 2015–2019 upGrad Education Private Limited.

CORPORATE

[Hire with upGrad](#)

[upGrad careers](#)

[upGrad xchange](#)

