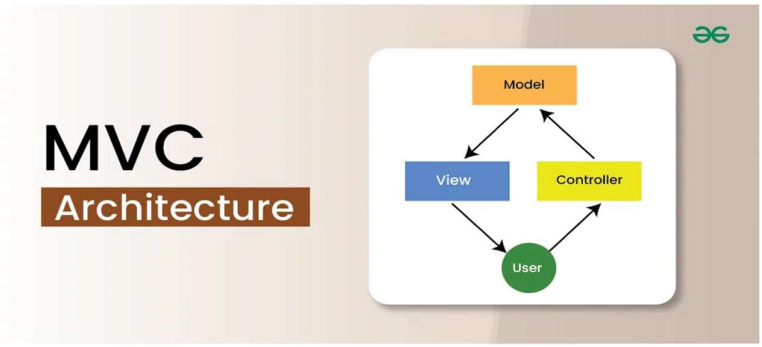
FULL STACK DEVELOPMENT

Unit-4

Imp Q & A

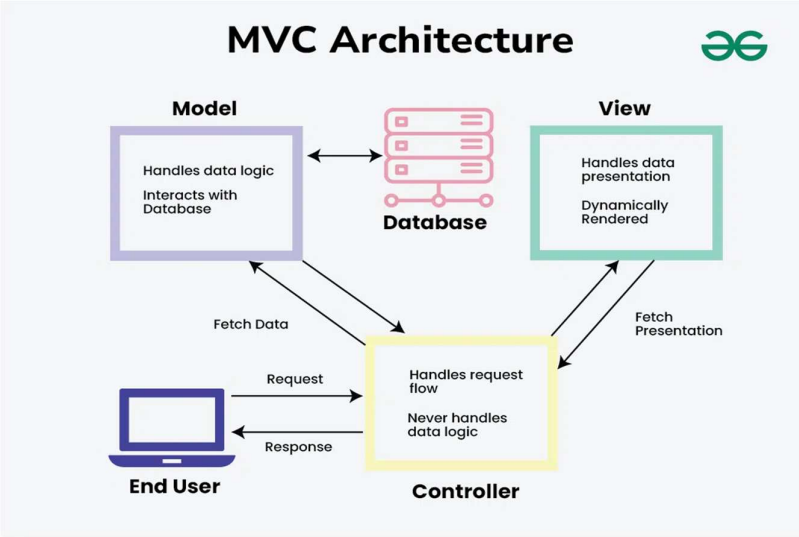
1) What is MVC architecture?

Ans) MVC(Model-View-Controller) Architecture is a fundamental design pattern in software development, separating an application into Model, View, and Controller components. This article explores its role in building robust, maintainable systems, emphasizing its benefits and implementation strategies.



What is MVC Architecture?

MVC (Model-View-Controller) architecture is a universal pattern of a structure in which an application is divided into three parts which are all dedicated to certain parts of the whole application. This pattern is normally used in software development to create organized and easy-to-maintain code. Here's a deeper look at each component:



- **Model:** It is worth stating that the Model stands as the data layer for the application. It is directly involved in managing the data as well as the control of the application's logic and rules.

- **View:** The View is in the presentation tier. It plays a role of presenting the information given by the Model to the user and transferring the user commands to the Controller. The View is used to display the data to the user in a readable and manageable way using the interface created by the Controller.
- **Controller:** The Controller CE works in the middle between the Model and the View. It takes the input from the View, sometimes modifies it with the help of the Model, and sends it back to the View. the results back to the View.

Importance in System Design

The MVC architecture is significant in system design for the following reasons:

- **Separation of Concerns:** MVC structures an application into three integrated elements and that really separates concern. Due to the clear division of responsibilities among each of the components, the functioning of the application becomes more logical and comprehensible.
- **Reusability:** Due to the fact that Model, View, and Controller are all the distinct entities, components can be utilized in the various sections of the application or different projects. For example, a Model class that contains user data can be used many times in the views and the controllers.
- **Scalability:** MVC amplifies the creation of applications that can be developed further. When the application advances, new functionalities of the application can be added without significant alterations to some parts of the application because they are elusive.
- **Testability:** This separation of concerns make it easier for the testing of each part from the other as we as from other problems. One of the testing strategies is to have separate tests for Model, View, and Controller parts, in this way, one is sure that each part is functioning properly before combining them.

History of MVC

MVC was introduced by Dr. Trygve Reenskaug into Smalltalk-76 programming language when he visited the Xerox Palo Alto Research Center (PARC) in mid-1970. Later the implementation became popular in other versions of Small-Talk. Then in the year 1988, the articles in "The Journal of Object Technology (JOT)" bring the whole picture of MVC as a well-accepted concept.

The different versions of MVC later came into existence with the requirement for application designing. These are:

- Hierarchical model-view-controller (HMVC).
- Model-view-presenter (MVP).
- Model-view-adapter (MVA).

- Model-view-viewmodel (MVVM) and some others.

Let us discuss the three components of MVC in brief:

1. Model: The Model encloses the clean application related data. But the model does not deal with any logic about how to present the data.
2. View: The View element is used for presenting the data of the model to the user. This element deals with how to link up with the model's data but doesn't provide any logic regarding what this data all about or how users can use these data.
3. Controller: The Controller is in between the model and the view element. It listens to all the incident and actions triggered in the view and performs an appropriate response back to the events.

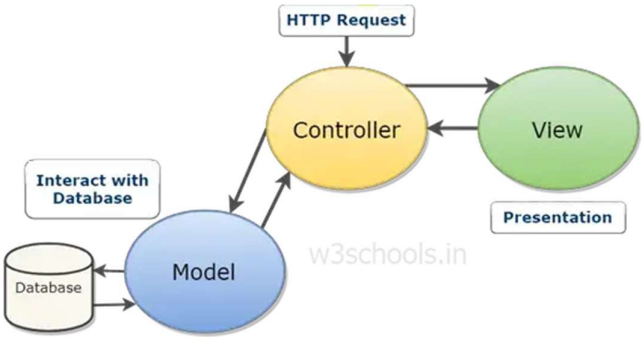


Fig: MVC Architecture

Benefits of MVC Architecture

MVC architecture offers several advantages:

- **Enhanced Organization**
- **Parallel Development**
- **Code Reusability**
- **Improved Maintainability**
- **Testability**
- **Implementation Strategies**

- Logical clustering of related acts on any controller can be achieved through MVC.
- Various developers can work at the same time on different parts the same application-controller, model, and the views part.
- In MVC, models can have numerous views.

Some of the real life examples of MVC architecture

Several popular web frameworks implement the MVC architecture:

- **Django:** A Python Web application framework that supports high level and promotes the strong, fast web application development and clear, and practical design. While Django also has the components of MVC but refers to it as MVT where T stands for Template, django is famous for its “batteries included” philosophy where it has many included features.
- **Ruby on Rails:** Ruby web application framework implemented on the server side. Rails is built on MVC and is a software that aims at having a number of conventions at the core, so that one can begin creating web applications at a very basic level.
- **Angular:** The libraries are created to build a single-page client application using HTML and a TypeScript language. Angular follows the MVC model by the use of components and dependency injection.

2) Create Restful API SIMPLE APPLICATION using Spring Framework?

Ans) Steps to Run

1. **Create a Spring Boot Project** using Spring Initializr
2. **Add Dependencies**: Spring Web
3. **Create Model, Service & Controller**
4. **Run and Test the API using Postman or Browser**

---

1. Project Structure

student-app

|— src/main/java/com/example/studentapp

| ├— model/Student.java

| ├— service/StudentService.java

| ├— controller/StudentController.java

| ├— StudentApplication.java

|— pom.xml

2. Code Implementation

Student Model (Student.java)

package com.example.studentapp.model;

```
public class Student {
    private int id;
    private String name;
    private String course;

    // Constructor
    public Student(int id, String name, String course) {
        this.id = id;
        this.name = name;
        this.course = course;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getCourse() { return course; }
    public void setCourse(String course) { this.course = course; }
}
```

Service (StudentService.java)

```
package com.example.studentapp.service;

import com.example.studentapp.model.Student;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.List;
```

```
@Service
public class StudentService {
    private final List<Student> students = new ArrayList<>();

    public List<Student> getAllStudents() {
        return students;
    }

    public Student getStudentById(int id) {
        return students.stream().filter(s -> s.getId() == id).findFirst().orElse(null);
    }

    public void addStudent(Student student) {
        students.add(student);
    }

    public void updateStudent(int id, Student updatedStudent) {
        students.forEach(s -> {
            if (s.getId() == id) {
                s.setName(updatedStudent.getName());
                s.setCourse(updatedStudent.getCourse());
            }
        });
    }

    public void deleteStudent(int id) {
        students.removeIf(s -> s.getId() == id);
    }
```

}

Controller (StudentController.java)

```
package com.example.studentapp.controller;

import com.example.studentapp.model.Student;
import com.example.studentapp.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/students")
public class StudentController {

    @Autowired
    private StudentService service;

    @GetMapping
    public List<Student> getAllStudents() {
        return service.getAllStudents();
    }

    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable int id) {
        return service.getStudentById(id);
    }

    @PostMapping
```

```
    public String addStudent(@RequestBody Student student) {
        service.addStudent(student);
        return "Student added!";
    }

    @PutMapping("/{id}")
    public String updateStudent(@PathVariable int id, @RequestBody Student student) {
        service.updateStudent(id, student);
        return "Student updated!";
    }

    @DeleteMapping("/{id}")
    public String deleteStudent(@PathVariable int id) {
        service.deleteStudent(id);
        return "Student deleted!";
    }
}
```

Main Application (StudentApplication.java)

```
package com.example.studentapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class StudentApplication {
    public static void main(String[] args) {
        SpringApplication.run(StudentApplication.class, args);
    }
}
```

3. Running & Testing

1. **Run** StudentApplication.java
2. **Test API in Postman or Browser**

HTTP Method	Endpoint	Description
GET	/students	Get all students
GET	/students/{id}	Get student by ID
POST	/students	Add a new student
PUT	/students/{id}	Update student
DELETE	/students/{id}	Delete student

Example JSON for POST /students

```
{
  "id": 1,
  "name": "Alice",
  "course": "Math"
}
```