MVC e MVP em JAVA

Arquitetura MVC em Java

ă INTRODUÇÃO

O Model-View-Controller (MVC) é um padrão de design

Bem conhecido no campo de desenvolvimento web. É uma maneira de organizar nosso código.

ă INTRODUÇÃO

O Model-View-Controller (MVC) é um padrão de design

Especifica que um programa ou aplicativo deve consistir em modelo de dados, informações de apresentação e informações de controle. O padrão MVC precisa que todos esses componentes sejam separados como objetos diferentes.

ă INTRODUÇÃO

Nesta aula, discutiremos a Arquitetura MVC em Java, juntamente com suas vantagens e desvantagens e exemplos para entender a implementação do MVC em Java.

O que é arquitetura MVC em Java?

Os projetos de modelo baseados na arquitetura MVC seguem o padrão de projeto MVC. A lógica do aplicativo é separada da interface do usuário ao projetar o software usando projetos de modelo.

MVC

A arquitetura do padrão MVC consiste em três camadas:

Modelo: Representa a camada de negócios da aplicação. É um objeto para transportar os dados que também pode conter a lógica para atualizar o controlador se os dados forem alterados.

MVC

View: Representa a camada de apresentação da aplicação. Ele é usado para visualizar os dados que o modelo contém.

MVC

Controlador: Funciona tanto no modelo quanto na visualização. Ele é usado para gerenciar o fluxo da aplicação, ou seja, o fluxo de dados no objeto do modelo e para atualizar a visualização sempre que os dados são alterados.

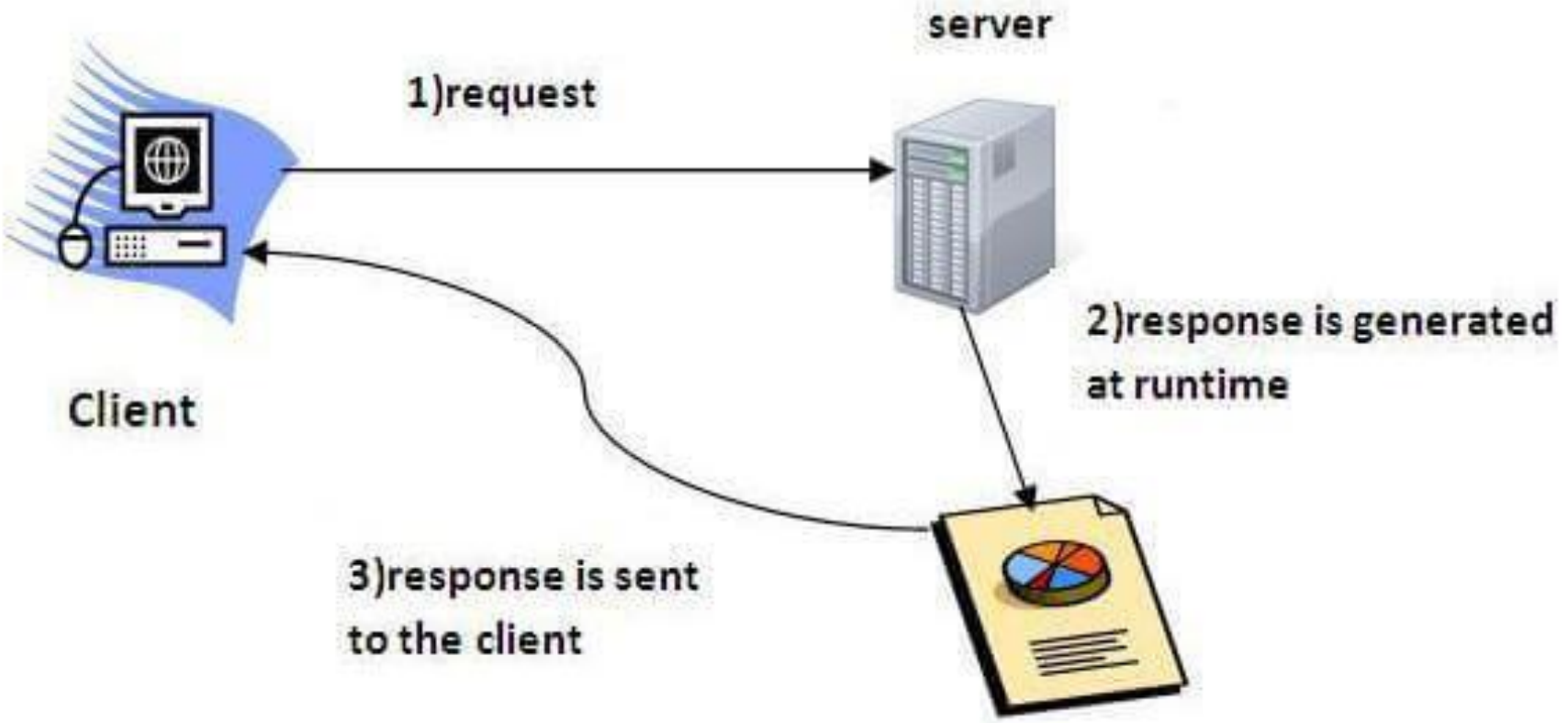
MVC

Na programação Java, o Model contém as classes Java simples, a View usada para exibir os dados e o Controller contém os servlets

ă SERVLET

Servlet é uma tecnologia que é usada para criar uma aplicação web, uma API que fornece muitas interfaces e classes, incluindo documentação. Pode ser descrito como uma classe que estende os recursos dos servidores e responde às solicitações recebidas. Ele pode responder a qualquer solicitação. Um componente da web que é implementado no servidor para criar uma página da web dinâmica.

SERVLET


server
1)request
2)response is generated
at runtime
Client
3)response is sent
to the client

ă MVC

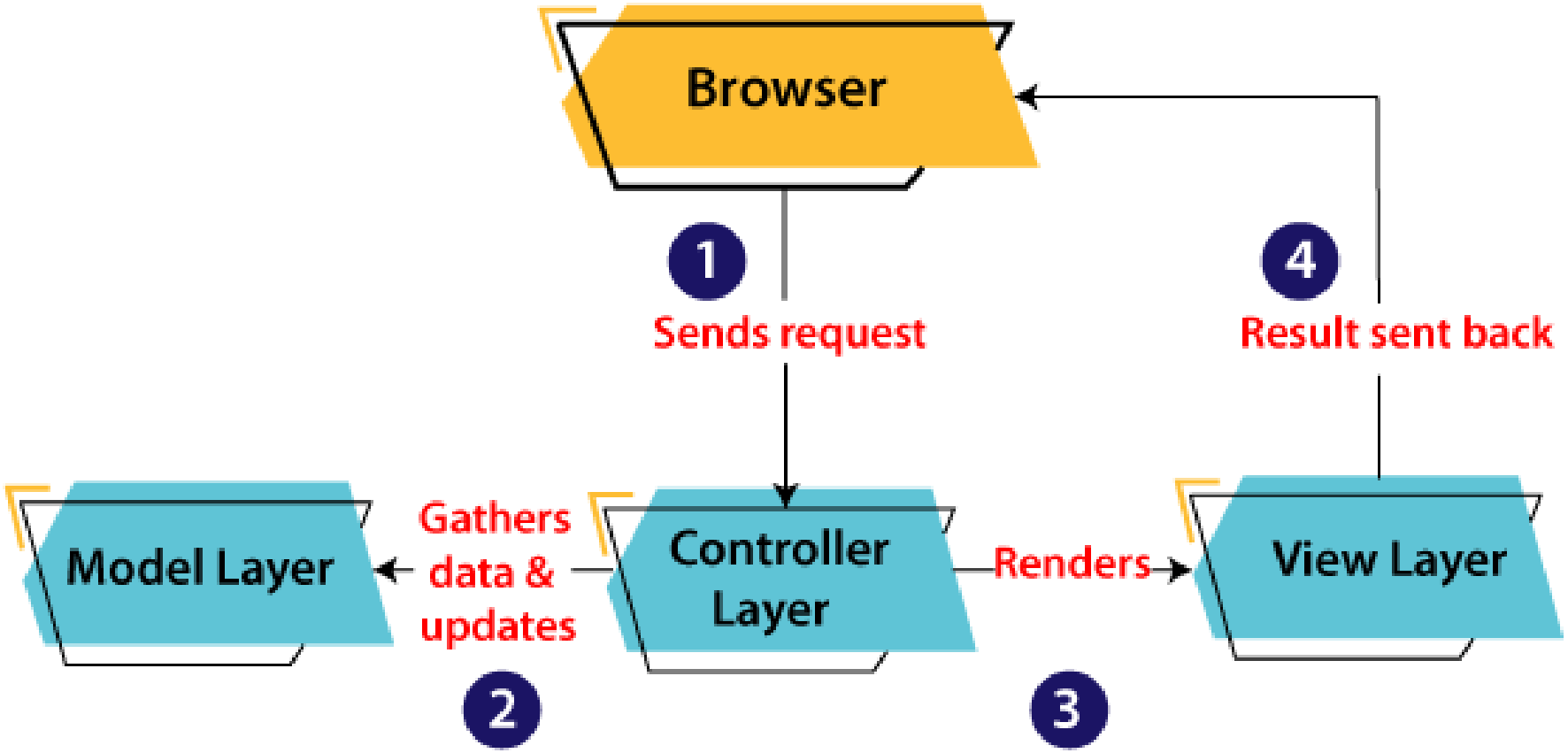
1. Um cliente (navegador) envia uma solicitação ao controlador no lado do servidor, para uma página.

2. O controlador então chama o modelo. Ele reúne os dados solicitados.

3. Em seguida, o controlador transfere os dados recuperados para a camada de visualização.

4. Agora o resultado é enviado de volta ao navegador (cliente) pela view.

MVC


Browser
1
Sends request
4
Result sent back
Model Layer
Gathers data & updates
2
Controller Layer
Renders
3
View Layer

ă MVC - Vantagens da arquitetura MVC

As vantagens da arquitetura MVC são as seguintes:

O MVC tem o recurso de escalabilidade que por sua vez ajuda no crescimento da aplicação.

Os componentes são fáceis de manter porque há menos dependência.

ă MVC - Vantagens da arquitetura MVC

As vantagens da arquitetura MVC são as seguintes:

Um modelo pode ser reutilizado por várias visualizações que fornecem reutilização de código.

Os desenvolvedores podem trabalhar com as três camadas (Model, View e Controller) simultaneamente.

ă MVC - Vantagens da arquitetura MVC

As vantagens da arquitetura MVC são as seguintes:

Usando MVC, o aplicativo se torna mais compreensível.

Usando MVC, cada camada é mantida separadamente, portanto, não precisamos lidar com código massivo.

A extensão e teste do aplicativo é mais fácil.

IMPLEMENTANDO COM JAVA

MVC

Para implementar o padrão MVC em Java, precisamos criar as três classes a seguir.

Classe Employee , atuará como camada de modelo

Classe EmployeeView , atuará como uma camada de visualização

Classe EmployeeContoller , atuará como uma camada de controlado

Camadas de Arquitetura MVC

Camada de modelo

O modelo no padrão de projeto MVC atua como uma camada de dados para o aplicativo. Ele representa a lógica de negócios para o aplicativo e também o estado do aplicativo.

MVC

O objeto de modelo busca e armazena o estado do modelo no banco de dados. Usando a camada de modelo, as regras são aplicadas aos dados que representam os conceitos de aplicação.

MVC

Vamos considerar o seguinte trecho de código que cria um que também é o primeiro passo para implementar o padrão MVC.

Funcionário.java

Parte 1

```
1.// class that represents model
2.public class Employee {
3.
4.     // declaring the variables
5.      private String EmployeeName;
6.      private String EmployeeId;
7.      private String EmployeeDepartment;
```

Funcionário.java

Parte 2

```
// defining getter and setter methods

1.      public String getId() {
2.        return EmployeeId;
3.      }
4.
5.      public void setId(String id) {
6.        this.EmployeeId = id;
7.      }
8.
9.      public String getName() {
10.         return EmployeeName;
11.       }
12.
13.       public void setName(String name) {
14.         this.EmployeeName = name;
15.       }
16.
17.       public String getDepartment() {
18.           return EmployeeDepartment;
19.         }
20.
21.       public void setDepartment(String Department) {
22.           this.EmployeeDepartment = Department;
23.         }
24.
25.   }
```

O código acima consiste simplesmente em métodos getter e setter para a classe Employee

Ver camada

Como o nome indica, view representa a visualização dos dados recebidos do modelo. A camada de visualização consiste na saída do aplicativo ou da interface do usuário. Ele envia os dados solicitados ao cliente, que são buscados na camada de modelo pelo controlador.

EmployeeView.java

```
// class which represents the view

1.public class EmployeeView {
2.
3.     // method to display the Employee details
4.public void printEmployeeDetails (String EmployeeName, Stri
ng EmployeeId, String EmployeeDepartment){
5.         System.out.println("Employee Details: ");
6.         System.out.println("Name: " + EmployeeName);
7.         System.out.println("Employee ID: " + EmployeeId);
8.         System.out.println("Employee Department: " + Employe
eDepartment);
9.     }
10.
```

Camada do Controlador

A camada controladora recebe as solicitações do usuário da camada de visualização e as processa, com as validações necessárias. Ele atua como uma interface entre Model e View. As solicitações são então enviadas ao modelo para processamento de dados. Depois de processados, os dados são enviados de volta ao controlador e exibidos na visualização.

EmployeeController .java

Parte 1

```
// class which represent the controller

1.public class EmployeeController {
2.
3.     // declaring the variables model and view
4.      private Employee model;
5.      private EmployeeView view;
6.
7.     // constructor to initialize
8.       public EmployeeController(Employee model, EmployeeVi
ew view) {
9.         this.model = model;
10.          this.view = view;
11.      }
```

EmployeeController .java

Parte 2

```
// getter and setter methods

1.    public void setEmployeeName(String name){
2.      model.setName(name);
3.    }
4.
5.    public String getEmployeeName(){
6.      return model.getName();
7.    }
8.
9.    public void setEmployeeId(String id){
10.       model.setId(id);
11.     }
12.
13.     public String getEmployeeId(){
14.       return model.getId();
15.     }
16.
17.     public void setEmployeeDepartment(String Department){
18.           model.setDepartment(Department);
19.     }
20.
21.       public String getEmployeeDepartment(){
22.           return model.getDepartment();
23.     }
```

EmployeeController .java

Parte 3

```
1.     // method to update view
2.      public void updateView() {
3.          view.printEmployeeDetails(model.getName(), model.ge
tId(), model.getDepartment());
4.      }
5.   }
```

Arquivo Java de classe principal

O exemplo a seguir exibe o arquivo principal para implementar a arquitetura MVC. Aqui, estamos usando a classe MVCMain.

MVCMain.java

Parte 1

```
1.// main class

1.public class MVCMain {
2.      public static void main(String[] args) {
3.
4.        // fetching the employee record based on the employee
_id from the database
5.        Employee model = retriveEmployeeFromDatabase();
6.
7.        // creating a view to write Employee details on console
8.        EmployeeView view = new EmployeeView();
9.
10.         EmployeeController controller = new EmployeeControl
ler(model, view);
11.
12.         controller.updateView();
```

MVCMain.java

Parte 2

```
1.        //updating the model data
2.        controller.setEmployeeName("Nirnay");
3.        System.out.println("\n Employee Details after updating:
 ");
4.
5.        controller.updateView();
6.      }
7.
8.      private static Employee retriveEmployeeFromDatabase(){
9.        Employee Employee = new Employee();
10.         Employee.setName("Anu");
11.         Employee.setId("11");
12.         Employee.setDepartment("Salesforce");
13.         return Employee;
14.       }
15.   }
```

Arquivo Java de classe principal

A classe **MVCMain** busca os dados do funcionário do método onde inserimos os valores. Em seguida, ele empurra esses valores no modelo. Depois disso, ele inicializa a visão (EmployeeView.java).

Arquivo Java de classe principal

Quando a visualização é inicializada, o Controlador (EmployeeController.java) é invocado e o vincula à classe Employee e à classe EmployeeView. Por fim, o método updateView() (método do controlador) atualiza os detalhes do funcionário a serem impressos no console.

Saída:

```
Detalhes do funcionário:
Nome: Ana
ID do funcionário: 11
Departamento de funcionários: Salesforce

Detalhes do funcionário após a atualização:
Nome: Nirnay
ID do funcionário: 11
Departamento de funcionários: Salesforce
```

Desta forma, aprendemos sobre a Arquitetura MVC, significado de cada camada e sua implementação em Java.

O Padrão MVP (Model-View-Presenter)

ă INTRODUÇÃO

O padrão MVP tem a finalidade de separar a camada de apresentação das camadas de dados e regras de negócio. Neste artigo vamos apresentar como usar o MVP em uma aplicação Delphi.

O MPV é divido em três partes bem distintas e com responsabilidades específicas, são elas o Model, View e Presenter.

ă INTRODUÇÃO

O padrão MVP tem a finalidade de separar a camada de apresentação das camadas de dados e regras de negócio. Neste artigo vamos apresentar como usar o MVP em uma aplicação Delphi.

O MPV é divido em três partes bem distintas e com responsabilidades específicas, são elas o Model, View e Presenter.

MVP

Em uma descrição simples podemos definir como:

View - em nosso caso é o formulário que exibirá os dados, não contém regra alguma do negócio a não ser disparar eventos que notificam mudança de estado dos dados que ele exibe e processamento próprio dele, como por exemplo código para fechar o formulário. Um objeto view implementa uma interface que expõe campos e eventos que o presenter necessita.

MVP

Em uma descrição simples podemos definir como:

Model - São os objetos que serão manipulados. Um objeto Model implementa uma interface que expõe os campos que o presenter irá atualizar quando sofrerem alteração na view.

MVP

MVP (Model — View — Presenter) entra em cena como uma alternativa ao padrão de arquitetura tradicional MVC (Model — View — Controller) .

Usando MVC como arquitetura de software, os desenvolvedores acabam com as seguintes dificuldades:

MVP - VANTAGENS

MVP (Model — View — Presenter)

A maior parte da lógica de negócios principal reside no Controller. Durante a vida útil de um aplicativo, esse arquivo cresce e torna-se difícil manter o código.

MVP - VANTAGENS

MVP (Model — View — Presenter)

Por causa da interface do usuário e dos mecanismos de acesso a dados fortemente acoplados, tanto a camada Controller quanto a View se enquadram na mesma atividade ou fragmento. Isso causa problema em fazer alterações nos recursos do aplicativo.

MVP - VANTAGENS

O padrão MVP supera esses desafios do MVC e fornece uma maneira fácil de estruturar os códigos do projeto. A razão pela qual o MVP é amplamente aceito é que ele fornece modularidade, testabilidade e uma base de código mais limpa e sustentável. É composto pelos três componentes seguintes:

MVP - VANTAGENS

Modelo Camada para armazenamento de dados. Ele é responsável por lidar com a lógica de domínio (regras de negócios do mundo real) e comunicação com o banco de dados e as camadas de rede.

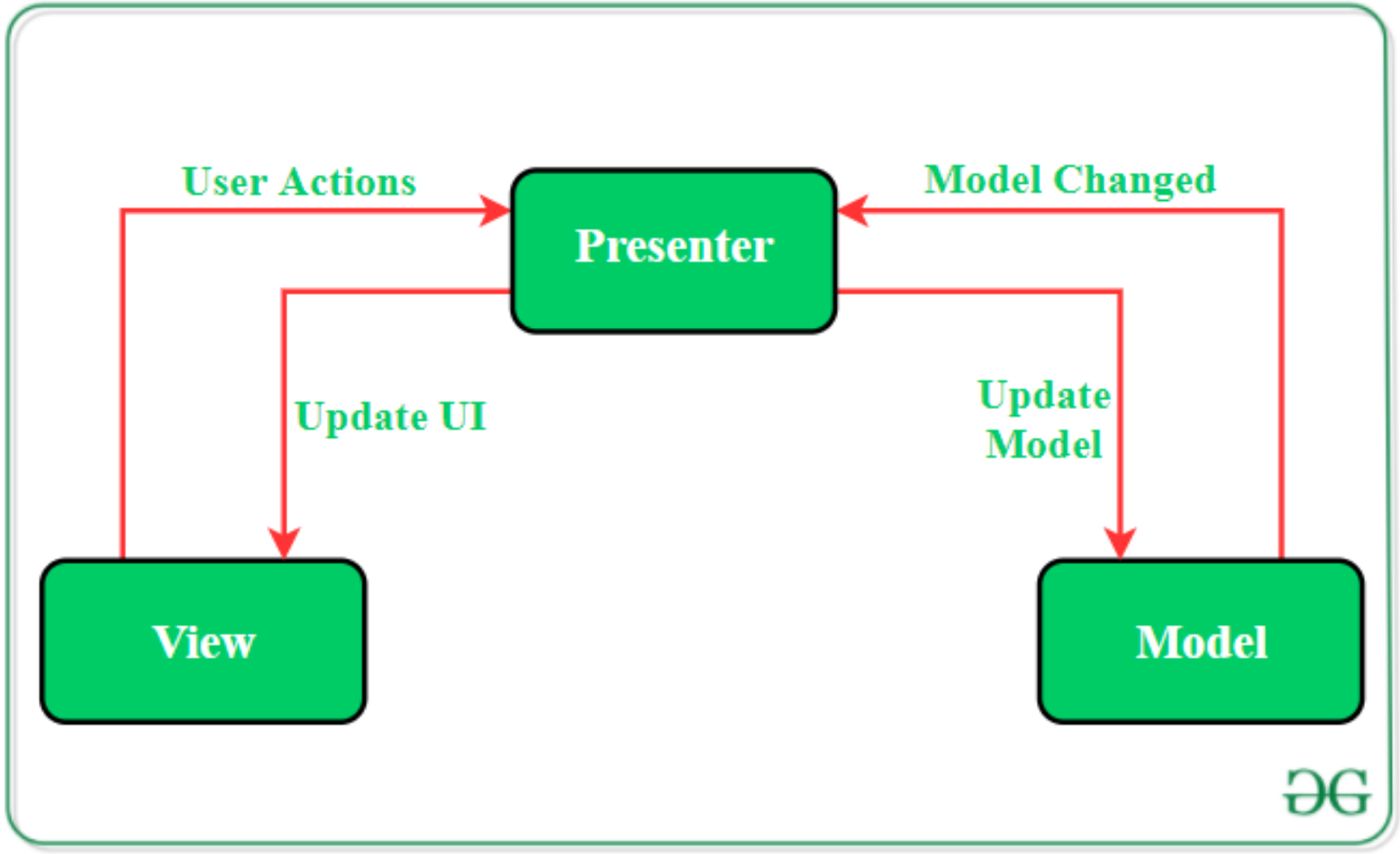
MVP - VANTAGENS

Visualização: camada UI (interface do usuário). Fornece a visualização dos dados e acompanha a ação do usuário para notificar o Apresentador.

MVP - VANTAGENS

Apresentador: busca os dados do modelo e aplica a lógica da interface do usuário para decidir o que exibir. Ele gerencia o estado da View e executa ações de acordo com a notificação de entrada do usuário da View.

MVP - VANTAGENS



MVP - Pontos-chave da arquitetura MVP

Pontos-chave da arquitetura MVP

A comunicação entre o View-Presenter e o Presenter-Model acontece por meio de uma interface (também chamada de Contrato) .

MVP - Pontos-chave da arquitetura MVP

Uma classe Presenter gerencia uma View por vez, ou seja, há um relacionamento um-para-um entre Presenter e View.

A classe Model e View não tem conhecimento sobre a existência uma da outra.

MVP - Vantagens e Desvantagens da arquitetura MVP

Vantagens: : Fácil manutenção e teste de código, pois o modelo, a visualização e a camada do apresentador são separados.

Desvantagens: Se o desenvolvedor não seguir o princípio de responsabilidade única para quebrar o código, a camada Presenter tende a se expandir para uma enorme classe onisciente.