MVC PATTERN

Qinshift Academy

WHAT IS MVC?

MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display.

This "separation of concerns" provides for a better division of labor and improved maintenance.

What is MVC?

Over the last few years, websites have shifted from simple HTML pages with a bit of CSS to incredibly complex applications with thousands of developers working on them at the same time.

To work with these complex web applications developers use different design patterns to lay out their projects, to make the code less complex and easier to work with.

The most popular of these patterns is MVC also known as **M**odel **V**iew **C**ontroller.

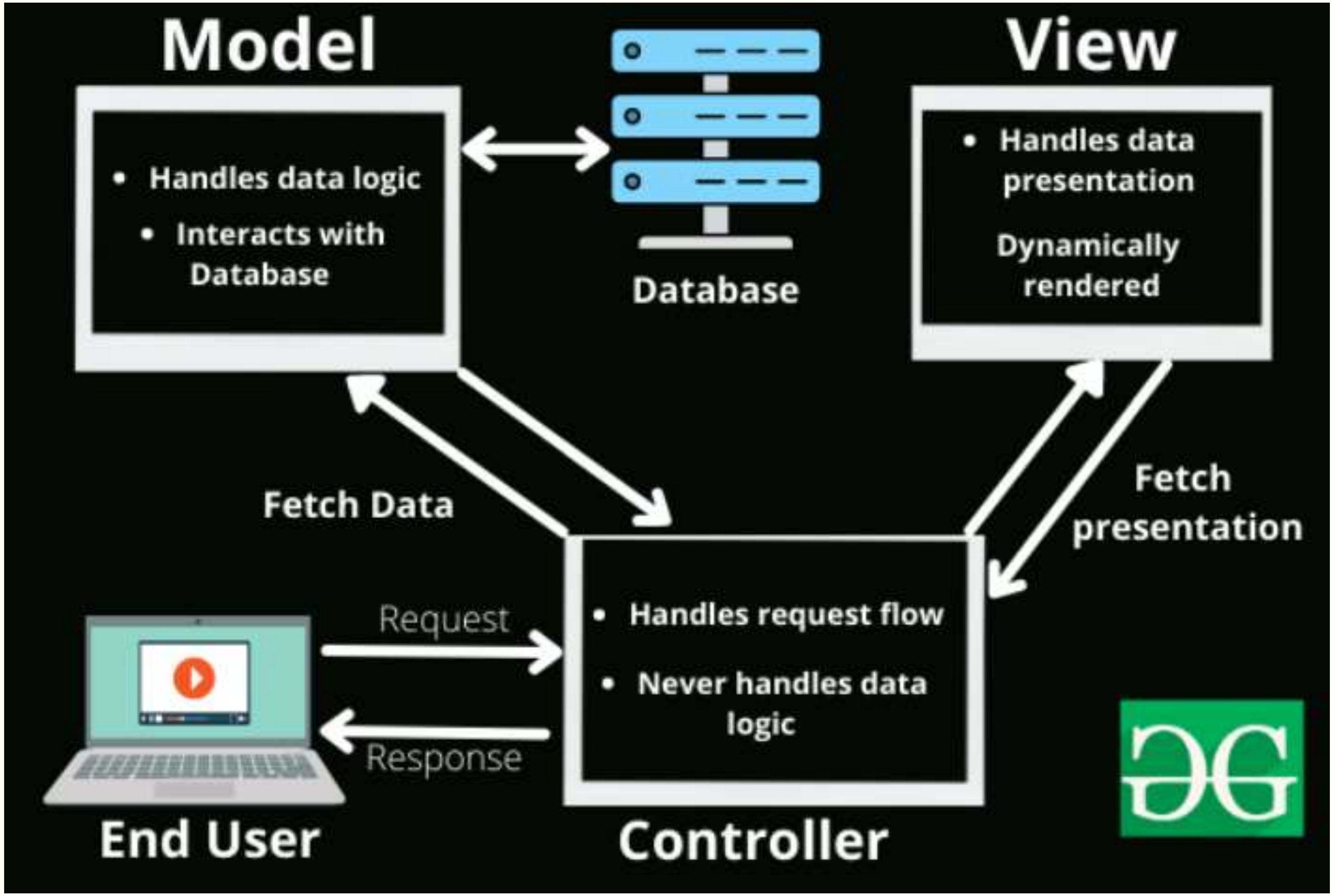
Features of MVC

- It provides a clear separation of business logic, UI logic, and input logic.
- It offers full control over your HTML and URLs which makes it easy to design web application architecture.
- It is a powerful URL-mapping component using which we can build applications that have comprehensible and searchable URLs.
- It supports Test Driven Development (TDD).

Components of MVC

The MVC framework includes the following 3 components:

- Controller
- Model
- View



Controller

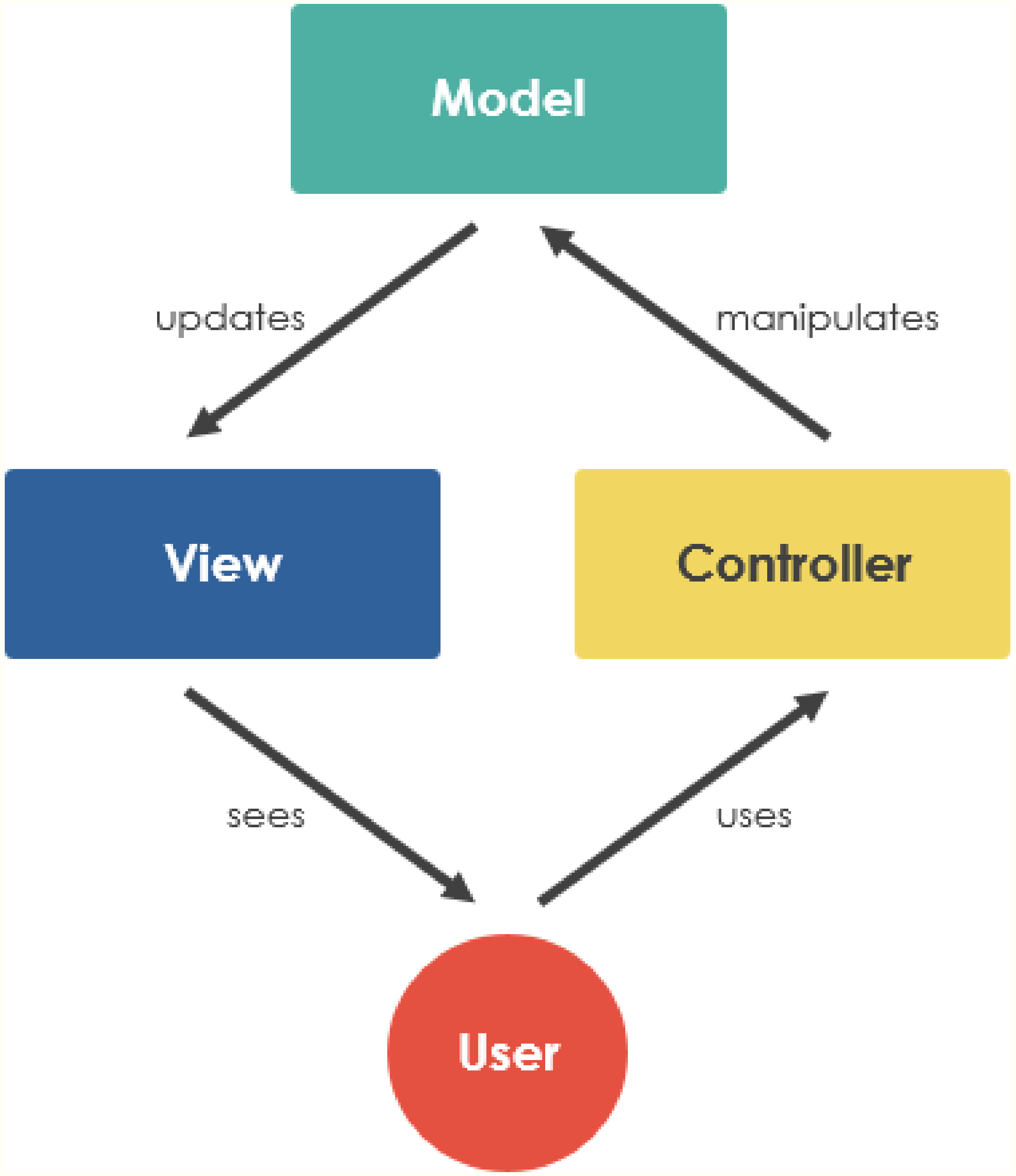
The **controller** is the component that enables the interconnection between the views and the model so it acts as an intermediary. The controller doesn’t have to worry about handling data logic, it just tells the model what to do. It process all the business logic and incoming requests, manipulate data using the **Model** component and interact with the **View** to render the final output.

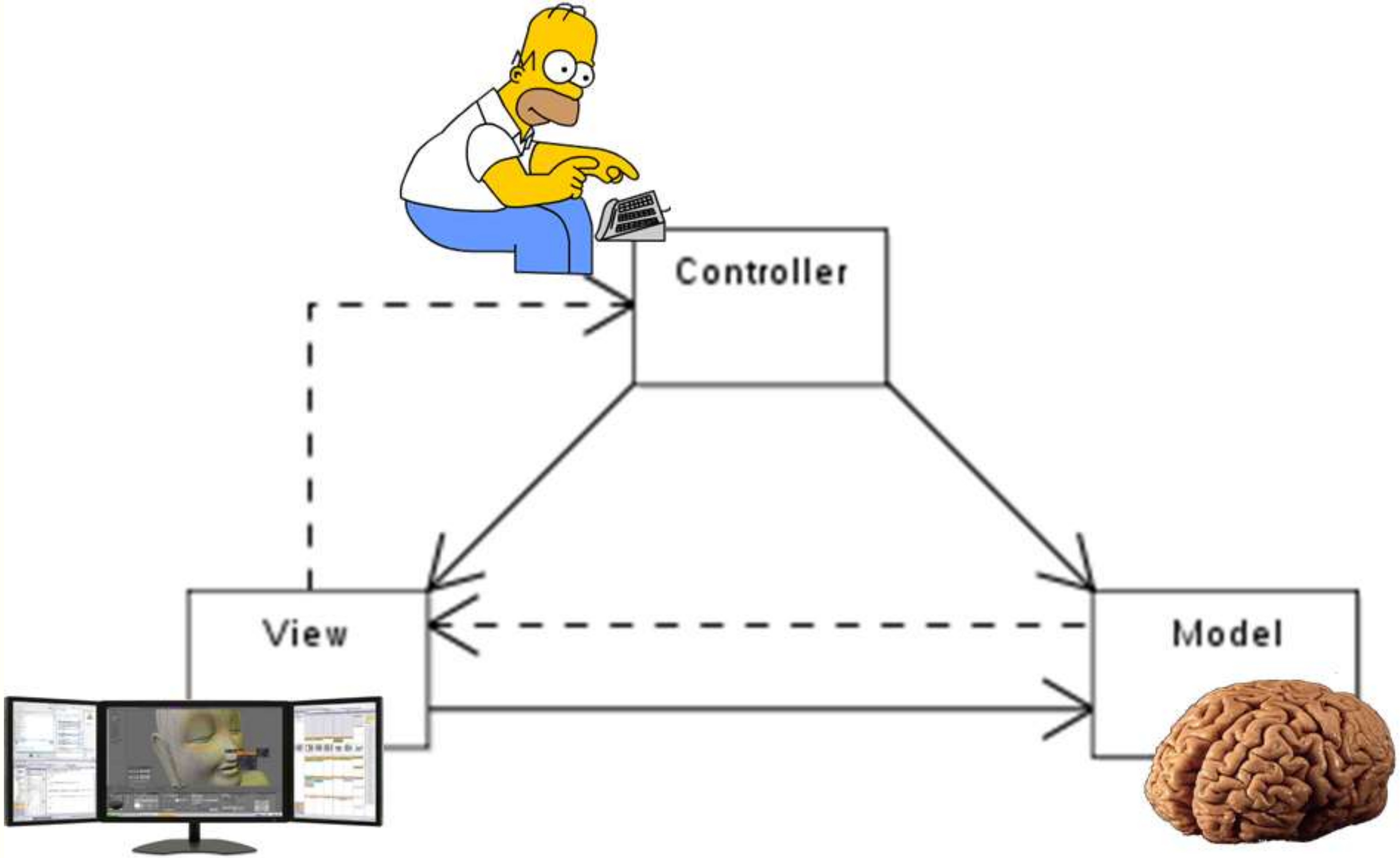
View

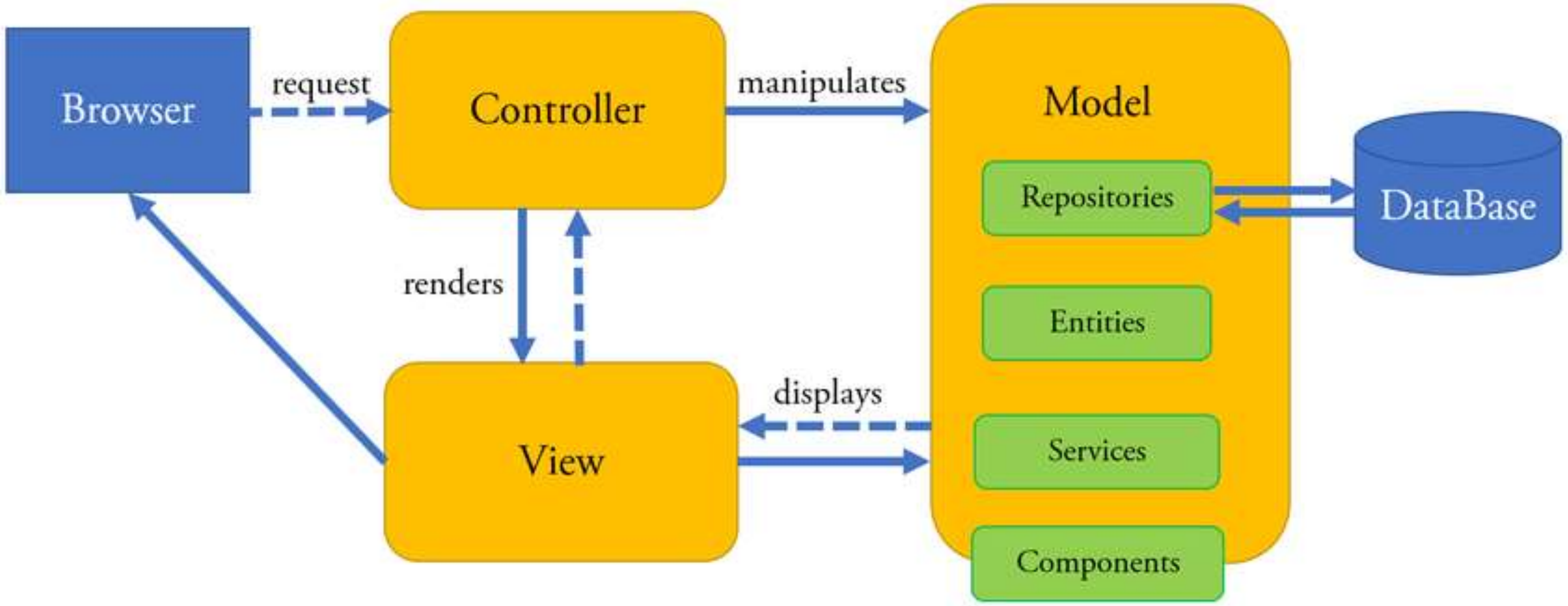
The **View** component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but these data aren’t taken directly but through the controller. It only interacts with the **controller**.

Model

The **Model** component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. It can add or retrieve data from the database. It responds to the controller’s request because the controller can’t interact with the database by itself. The model interacts with the database and gives the required data back to the controller.


Model
updates
manipulates
View
Controller
sees
uses
User


Controller
View
Model


Browser
request
Controller
manipulates
Model
Repositories
DataBase
renders
Entities
displays
View
Services
Components

Let’s understand the working of the MVC framework with an example

Let’s imagine an end-user sends a request to a server to get a list of students studying in a class. The server would then send that request to that particular controller that handles students. That controller would then request the model that handles students to return a list of all students studying in a class.

Let’s understand the working of the MVC framework with an example

Let’s imagine an end-user sends a request to a server to get a list of students studying in a class. The server would then send that request to that particular controller that handles students. That controller would then request the model that handles students to return a list of all students studying in a class.

Let’s understand the working of the MVC framework with an example

The model would query the database for the list of all students and then return that list back to the controller. If the response back from the model was successful, then the controller would ask the view associated with students to return a presentation of the list of students. This view would take the list of students from the controller and render the list into HTML that can be used by the browser.

Let’s understand the working of the MVC framework with an example

The controller would then take that presentation and returns it back to the user. Thus ending the request. If earlier the model returned an error, the controller would handle that error by asking the view that handles errors to render a presentation for that particular error. That error presentation would then be returned to the user instead of the student list presentation.

Let’s understand the working of the MVC framework with an example

As we can see from the above example, the model handles all of the data. The view handles all of the presentations and the controller just tells the model and view of what to do. This is the basic architecture and working of the MVC framework.

The MVC architectural pattern allows us to adhere to the following design principles:

Let’s understand the working of the MVC framework with an example

1. **Divide and conquer**: The three components can be somewhat independently designed.
2. **Increase cohesion**: The components have stronger layer cohesion than if the view and controller were together in a single UI layer.
3. **Reduce coupling**: The communication channels between the three components are minimal and easy to find.
4. **Increase reuse**: The view and controller normally make extensive use of reusable components for various kinds of UI controls. The UI, however will become application specific, therefore it will not be easily reusable
5. **Design for flexibility**: It is usually quite easy to change the UI by changing the view, the controller, or both.

MVC in backend applications

MVC on the back end means that whenever a http request comes in the Controller handles the operation by handing over the received data and communicating with the model. After it coordinates the operation typically it responds to the client by sending down a view. A view usually consists of some script that combines the view model with HTML. The view engine works the magic to generate HTML from this script. The script can be in the form of JSP, Razor scripts, PHP etc. The controller sends back that HTML, well not directly since there is usually an entire framework that handles the response details but to you as a user of that MVC framework it appears the controller sends it. All these operations take place on the back end.

QUESTIONS?

You can find us at:

mail

main

Qinshift Academy