# 2CSDE86 Application Development Frameworks (ADF)

Lecture-2

MVC, MVT

7th CSE

Daiwat Vyas & Ajaykumar Patel

# Disclaimer

- Some of the content in the ppt is taken from various online sources and reference books and after referring it was considered for including it in the ppt slides.

# Traditional Web Application Development

- **How traditional web application development worked?**

# Traditional Web Application Development

- **How traditional web application development worked?**

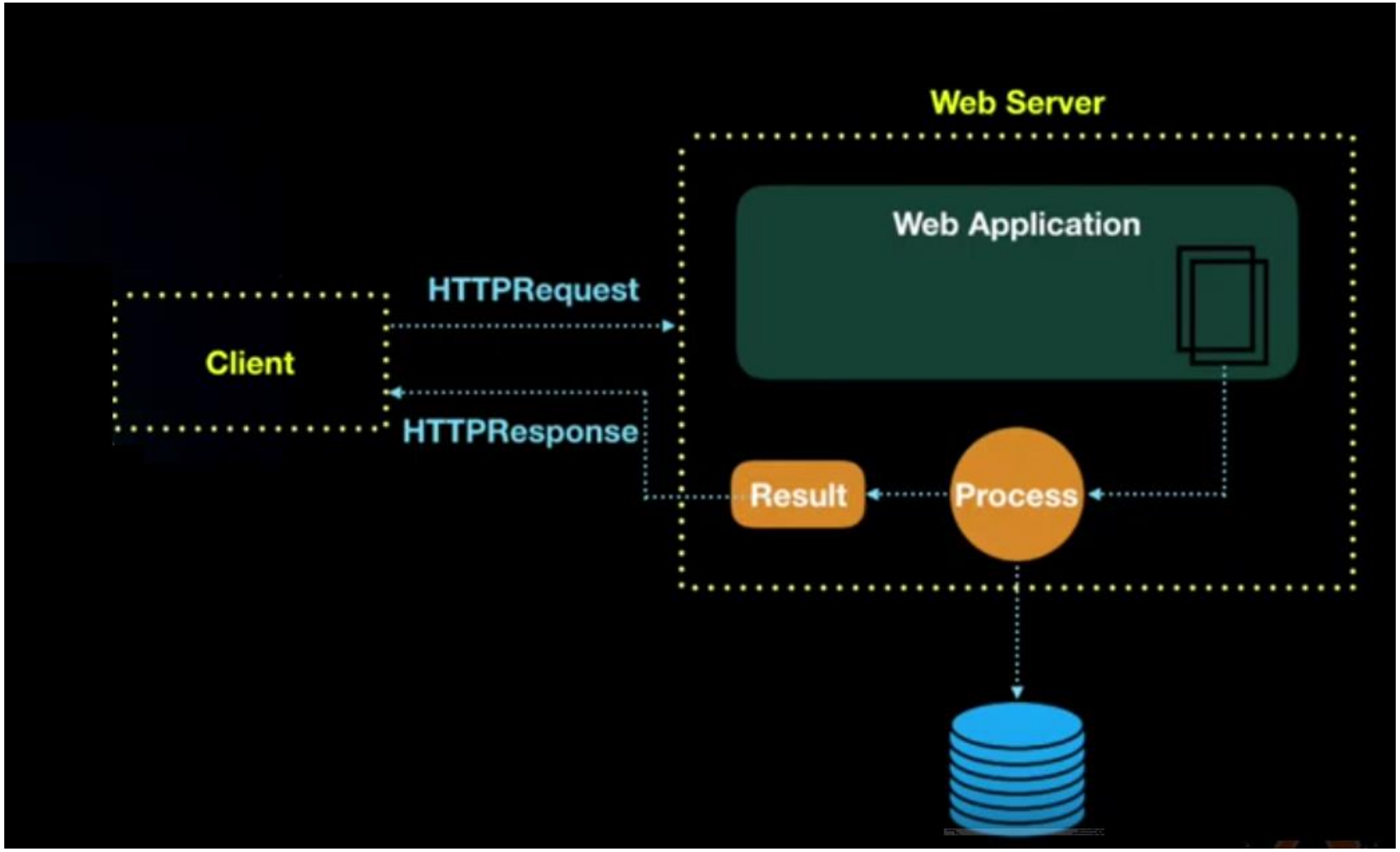


Figure:-1 Traditional Web Development Work-Flow

# Traditional Web Application Development

- **How traditional web application development worked?**

- Client sends request to web-server. The request will be given as a HttpRequest to the web-server.

- Server accepts the request and identifies the requester web application and then the requested page will be processed within the web-server. (It can reject the request too).

- If there is any requirement to interact with the database, then the process interacts with the database & generate the result which will be provided as a HttpResponse to the requester client.

# **M**odel-**V**iew-**C**ontroller (MVC)

- **History of MVC**

- Tygrve Reenskaug invented MVC. The first reports on MVC were written when he was visiting scientist at Xerox Palo Alto Research Laboratory (PARC) in 1978/79.

- At first, MVC was called “Thing Model View Editor” but rapidly changed it to “ Model View Controller.

- The goal of Tygrve was to solve the problem of users controlling a large and complex data set. The practice of MVC has changed over the years. Since the MVC pattern was invented before web browsers, initially was used as an architectural pattern for graphical user interfaces(GUI).

# **M**odel-**V**iew-**C**ontroller (MVC)

- **History of MVC (contd…)**

- Currently MVC is used for designing web applications.

- Some web frameworks that use MVC concept:

- Ruby on Rails
- Laravel
- CherryPy
- Symphony, etc

# **M**odel-**V**iew-**C**ontroller (MVC)

- What is MVC?

- MVC is an architectural design pattern that separated an application into three logical components:

  - Model

  - View

  - Component

# **M**odel-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

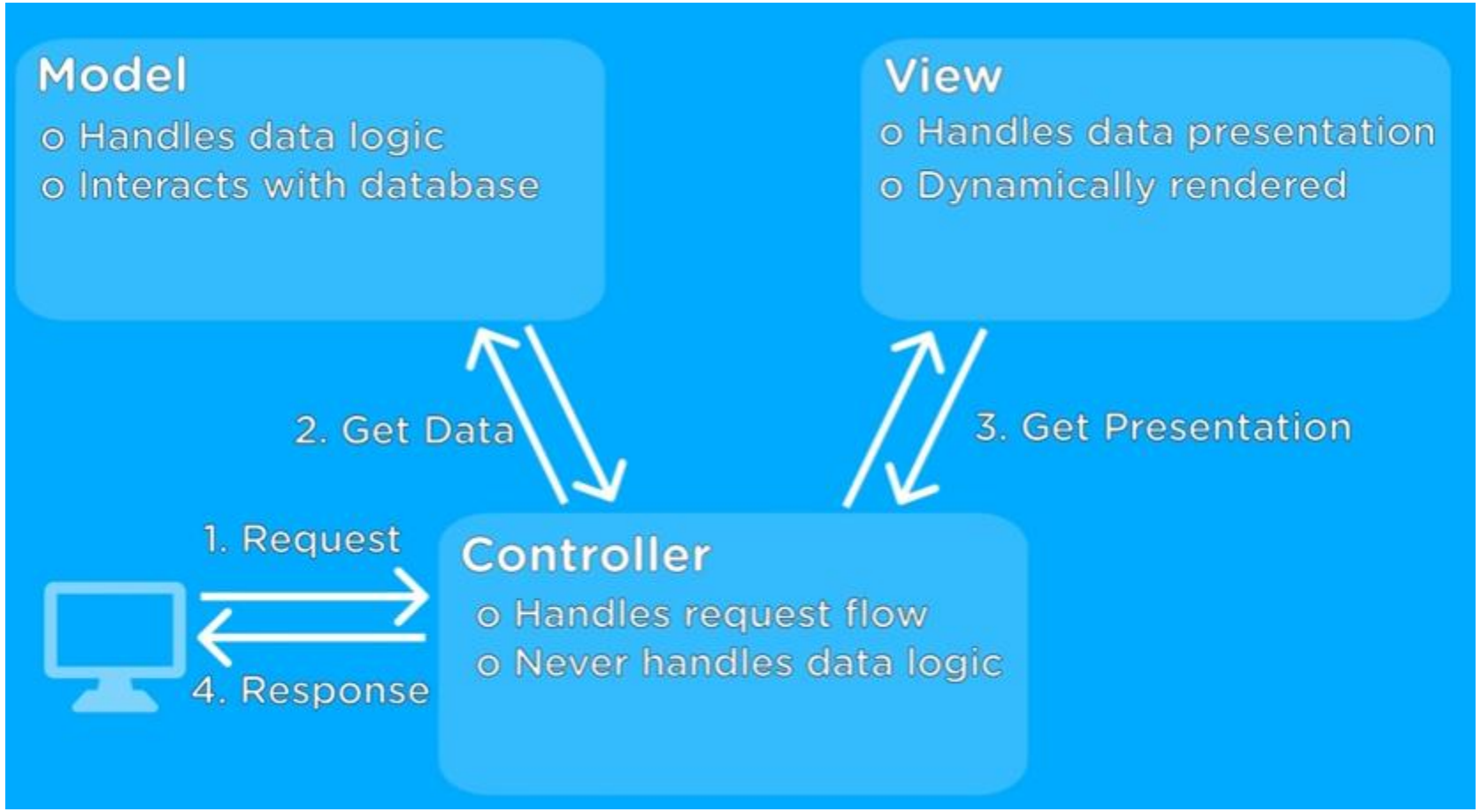


Figure:-2 MVC Architecture Representation

# **M**odel-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- What are the key points that you observe by referring the MVC architecture representation in the Figure:-2 in previous slide?

# **M**odel-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- What are the key points that you observe by referring the MVC architecture representation in the Figure:-2 in previous slide?

- Model and View never interact with each other directly.

- Any interactions between the Model and View happens through the Controller.

- What can be the advantage of having the Controller between the Model and View?

# **M**odel-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- What can be the advantage of having the Controller between the Model and View?

- Presentation of data and logic of data are completely separated.

- Helps in creating complex applications more easily.

# Model-View-Controller (MVC)

- What is MVC? Contd…

- Example

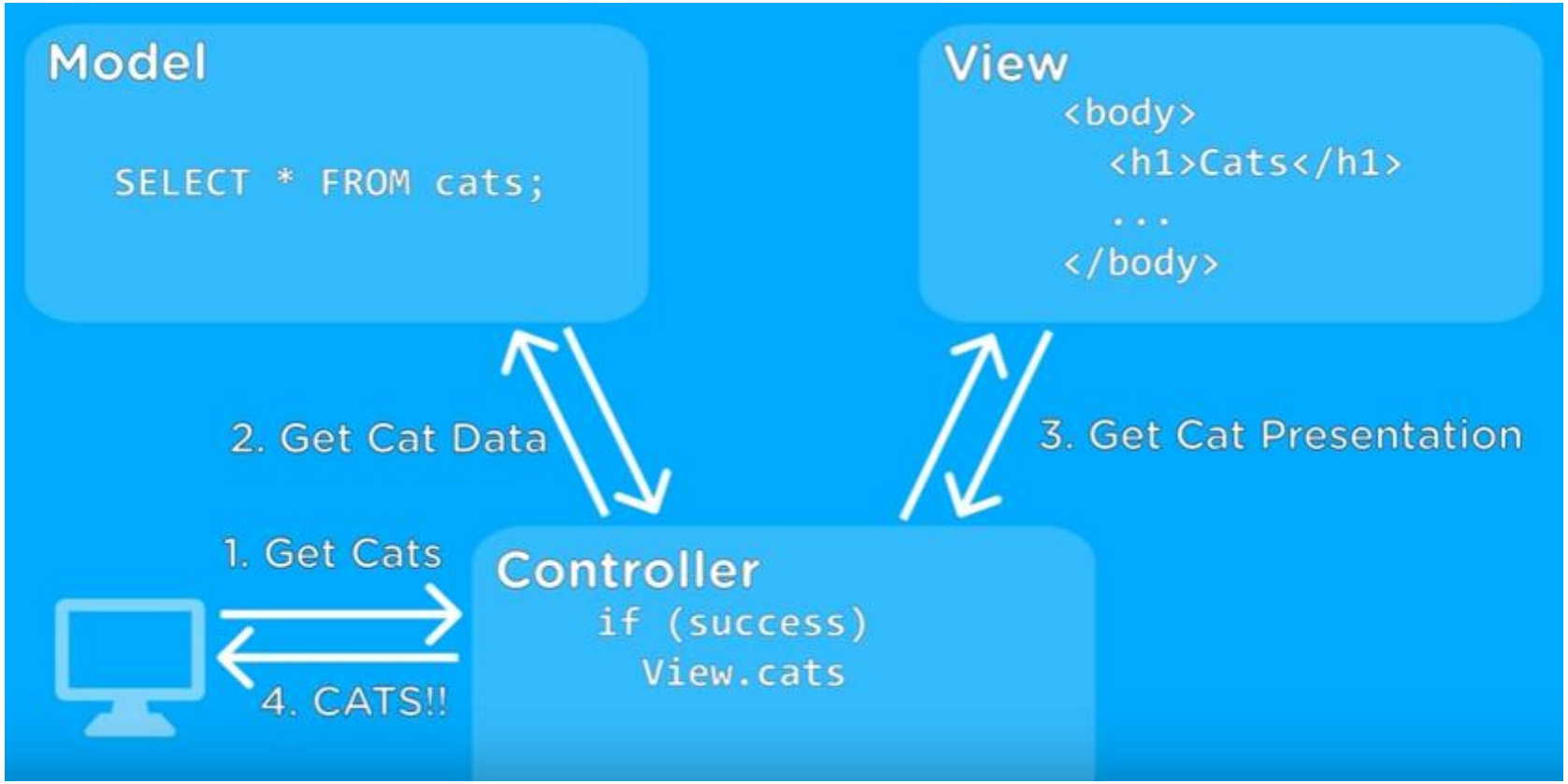


Figure:-3 Example for understanding MVC

# Model-View-Controller (MVC)

- What is MVC? Contd…
- In context to previous example, What if the model returns an error?

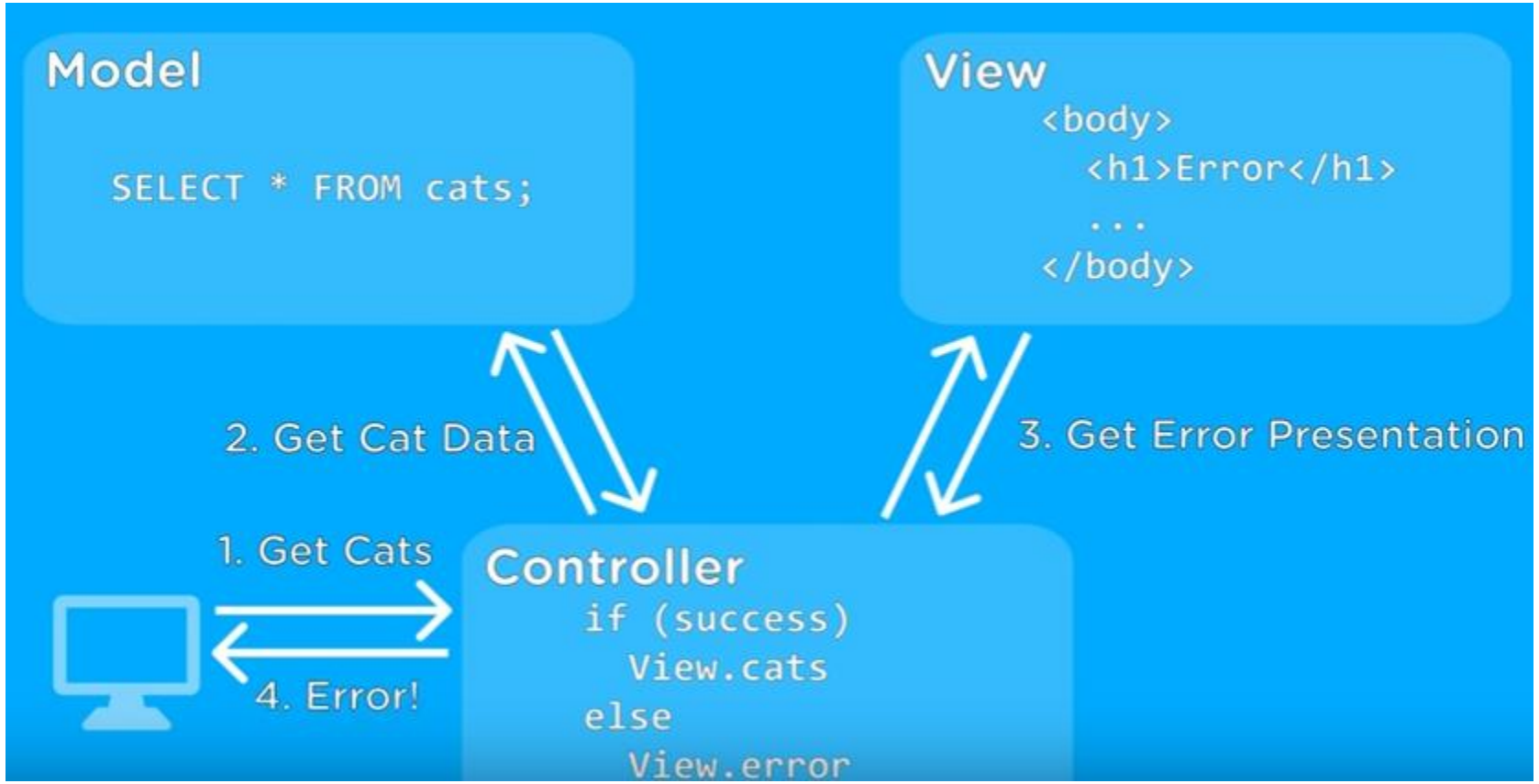


Figure:-4 Example for understanding how error is handled

# **M**odel-**V**iew-**C**ontroller (MVC)

- **What is MVC? Contd…**
- **Model:-**

- It is known as the lowest level which means it is responsible for maintaining data.
- Handle data logically so it basically deals with data.
- The model is actually connected to the database so anything you do with data.
- Adding or retrieving data is done in the model component.
- It responds to the controller requests because the controller never talks to the database by itself.
- The model talks to the database back and forth and then it gives the needed data to the controller. **Note: the model never communicates with the view directly.**

# **M**odel-**V**iew-**C**ontroller (MVC)

- **What is MVC? Contd…**
- **View:-**

- Data representation is done by the view component.
- It actually generates UI or user interface for the user.
- So at web applications when you think of the view component just think the Html/CSS part.
- Views are created by the data which is collected by the model component
- but **these data aren’t taken directly but through the controller, so the view only speaks to the controller.**

# **M**odel-**V**iew-**C**ontroller (MVC)

- **What is MVC? Contd…**
- **Controller:-**

- It is known as the main component because 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.
- After receiving data from the model it processes it and then it takes all that information it sends it to the view and explains how to represent to the user. **Note: Views and Models can not talk directly.**

# **M**odel-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- In context to an application using MVC architecture:

- Model – Handles all the Data

- View- Handles all the Presentation

- Controller- Tells the Model and View What to do

# **M**odel-**V**iew-**C**ontroller (MVC)

- To summarize MVC:

- The **Model** contains only the pure application data, it contains no logic describing how to present the data to a user.

- The **View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.

- The **Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

# **M**odel-**V**iew-**C**ontroller (MVC)

- To summarize MVC: (contd…) **key points**

- MVC is a popular way of organizing your code.
- The big idea behind MVC is that each section of your code has a purpose, and those purposes are different.

- Some of your code holds the data of your app, some of your code makes your app look nice, and some of your code controls how your app functions.

- MVC is a way to organize your code's core functions into their own, neatly organized boxes.
- This makes thinking about your app, revisiting your app, and sharing your app with others much easier and cleaner.

# **M**odel-**V**iew-**C**ontroller (MVC)

- **Analogy to understand MVC**

- MVC is a way to think about how an web application works.
- It’s kind of like how you prepare dinner for your party. **You have a fridge full of food, which is like the Model.** The fridge (Model) contains the raw materials we will use to make dinner.
- You also probably have a recipe or two. **A recipe (assuming you follow it exactly) is like the Controller of dinner you are preparing for your party**. Recipes dictate which stuff in the fridge you’ll take out, how you’ll put it together, and how long you need to cook it.
- Then, you have table-settings, silverware, dinner set etc., which are what your hungry friends and family use to eat dinner. **Table-top items are like the View. They let your guests interact with your Model and Controller’s creation.**

# **M**odel-**V**iew-**C**ontroller (MVC)

- **Another Analogy to understand MVC (contd…)**

- Let's assume you go to a restaurant. You will not go to the kitchen and prepare food which you can surely do at your home. Instead, you just go there and wait for the waiter to come on.
- Now the waiter comes to you, and you just order the food. The waiter doesn't know who you are and what you want he just written down the detail of your food order.
- Then, the waiter moves to the kitchen. In the kitchen waiter not prepare your food.
- The cook prepares your food. The waiter is given your order to him along with your table number.
- Cook then prepares food for you. He uses ingredients to cook the food. Let's assume that your order was for vegetable sandwich. Then he needs bread, tomato, potato, capsicum, onion, bit, cheese, etc. which he sources from the refrigerator.
- Cook finally hand’s over the food to the waiter. Now it is the job of the waiter to move this food outside the kitchen.
- Now waiter knows which food you have ordered and how they are served.

# **M**odel-**V**iew-**C**ontroller (MVC)

- **Another Analogy to understand MVC (contd…)**

- In the previous analogy:

- View= You
- Waiter= Controller
- Cook= Model
- Refrigerator= Data

# **M**odel-**V**iew-**C**ontroller (MVC)

- **Can you give a real life example application which can use MVC?**

# **M**odel-**V**iew-**C**ontroller (MVC)

**Real life example to understand MVC**

- MVC is helpful when planning your app, because it gives you an outline of how your ideas should be organized into actual code.
- For instance, let's imagine **you're creating a To-do list app**. This app will let users create tasks and organize them into lists.
- The **Model** in a To-Do app might define what a "task" is and that a "list" is a collection of tasks.
- The **View** code will define what the To-Do's and lists looks like, visually. The tasks could have large font, or be a certain color.
- Finally, the **Controller** could define how a user adds a task, or marks another as complete. The Controller connects the View's add button to the Model, so that when you click "add task," the Model adds a new task.

# **M**odel-**V**iew-**C**ontroller (MVC)

- Let us take an example in Java, to understand MVC:

- MVCPattern.java

# Model-View-Controller (MVC)

```
class Student
{
    private String rollNo;
    private String name;
    public String getRollNo()
    {
        return rollNo;
    }
    public void setRollNo(String rollNo)
    {
        this.rollNo = rollNo;
    }

    public String getName()
    {
        return name;   }
```

# Model-View-Controller (MVC)

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

# Model-View-Controller (MVC)

```
class StudentController
{
    private Student model;
    private StudentView view;

    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();
    }
```

# Model-View-Controller (MVC)

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

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

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

# Model-View-Controller (MVC)

```
class MVCPattern

{

    public static void main(String[] args)

    {

        Student model = retriveStudentFromDatabase();

        StudentView view = new StudentView();

        StudentController controller = new StudentController(model, view);

        controller.updateView();

        controller.setStudentName("Dvijesh Bhatt");



        controller.updateView();

    }
```

# **M**odel-**V**iew-**C**ontroller (MVC)

- **Just to conclude:**

- MVC is a framework for thinking about programming, and for organizing your program's files.

- To signify the idea that your code should be organized by its function, developers will create folders for each part of MVC.

- The idea that apps should be divided based on the *function* of each part of the code is sometimes referred to as *separation of concerns.*

# **M**odel-**V**iew-**C**ontroller (MVC)

- **Just to conclude: (contd…)**

- MVC gives you a starting place to translate your ideas into code, and it also makes coming back to your code easier, since you will be able to identify which code does what.

- In addition, the organizational standard MVC promotes makes it easy for other developers to understand your code.

# **M**odel-**V**iew-**C**ontroller (MVC)

- **Just to conclude: (contd…)**

- Thinking about how code interacts with other code is a significant part of programming, and learning to collaborate with other developers is an important skill.

- Taking the time to think about how your app fits into the MVC framework will level-up your skills as a developer by teaching you both.

# Post Lecture Tasks

- **Identify, why MVC was introduced to replace traditional application development model?**
- **List out advantages and dis-advantages of MVC architecture.**
- **Compare & Contrast 3-Tier architecture and MVC architecture.**

# **M**odel-**V**iew-**T**emplate (MVT)

- **What is MVT?**

- **M-** Model

- **V-** View

- **T-** Template

- Django prefers to use its own logic implementation in its web app and hence its framework handles all the controller parts by itself.

# **M**odel-**V**iew-**T**emplate (MVT)

- **What is MVT? (contd…)**

- **Model:-** It is same as in MVC, here as well it has the same functionality of providing the interface for the data stored in the database**.**

- **View:-** In Django, Views act as a link between the **Model data** and the **Templates.**

  - Note: Just like the controller in MVC, views in Django MVT are responsible for handling all the business logic behind the web app. It acts as a bridge between Models and Templates
  - It sees the user request, retrieves appropriate data from the database, then renders back the template along with retrieved data.
  - Therefore there is no separate controller in Django MVT architecture and everything is based on **Model -View – Template** itself and hence the name **MVT**.

# **M**odel-**V**iew-**T**emplate (MVT)

- **What is MVT? (contd…)**

- **Template:- Just as View** in MVC, Django uses templates in its framework. Templates are responsible for the entire User Interface. It handles all the static parts of the webpage along with the HTML, which the users visiting the webpage will perceive.

# **M**odel-**V**iew-**T**emplate (MVT)

- **What is MVT? (contd…)**

- **Template:- Just as View** in MVC, Django uses templates in its framework. Templates are responsible for the entire User Interface.

- It handles all the static parts of the webpage along with the HTML, which the users visiting the webpage will perceive.

- Templates are used to specify a structure for an output.
- Data can be populated in a template using placeholders. It defines how the data is presented. An example is a Generic list view that we can use to display a set of records from the database.

# Model-View-Template (MVT)

- **Difference between MVC and MVT**

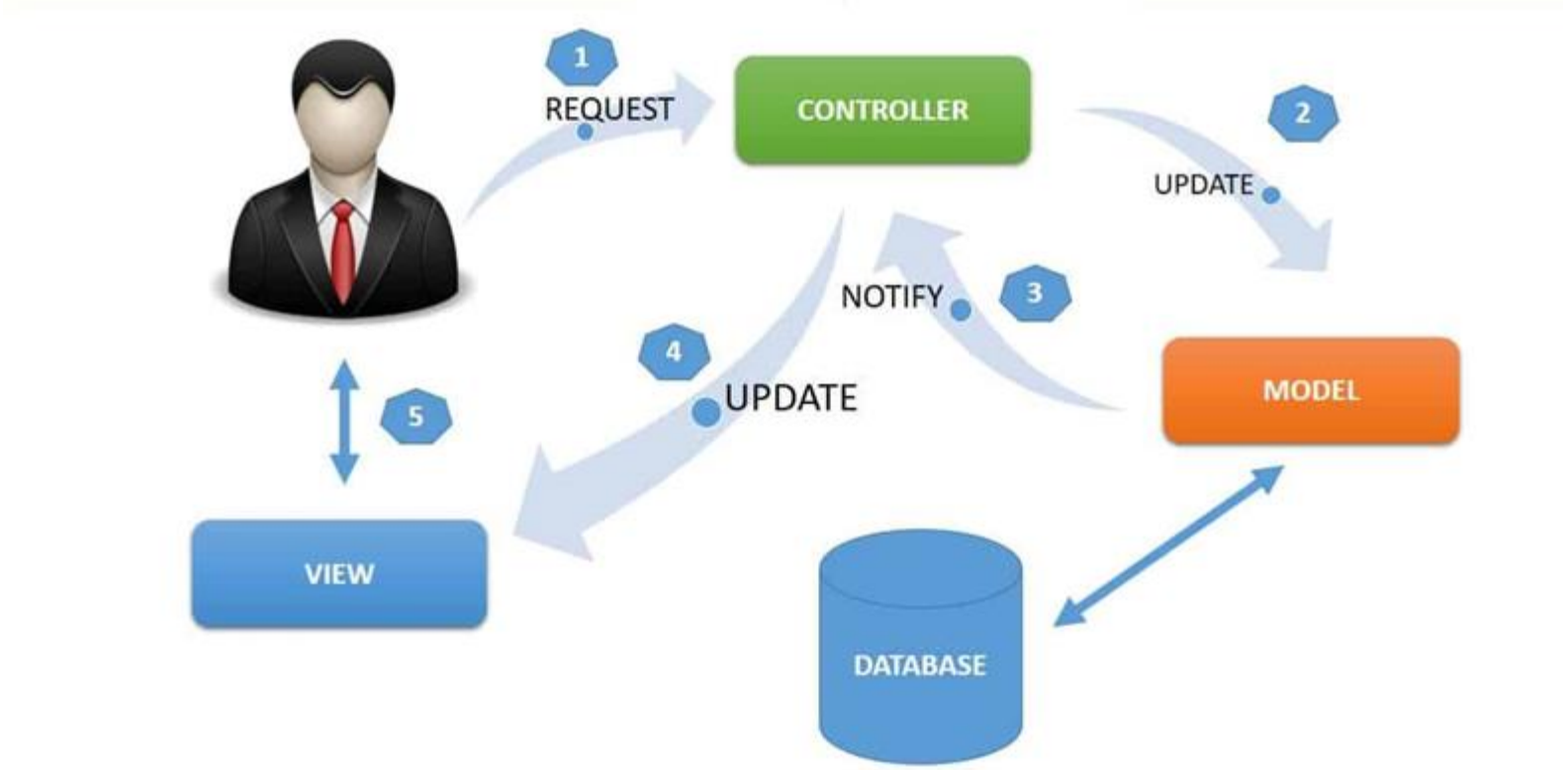


Figure:-5 MVC Architecture

# Model-View-Template (MVT)

- **Difference between MVC and MVT**

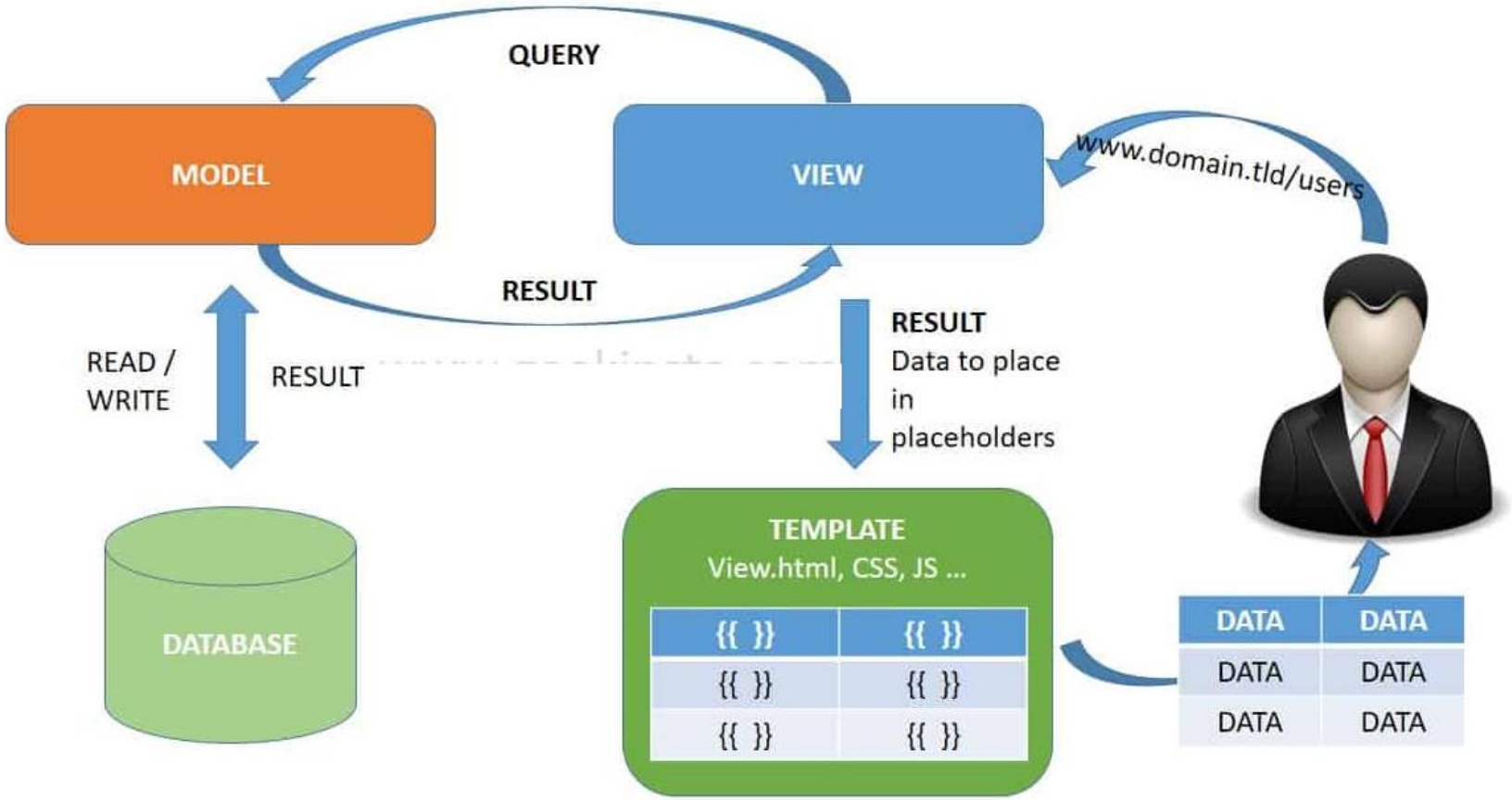


Figure:-6 MVT Architecture

# **M**odel-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT**

- The main difference between MVC and MVT is that in a Model View Controller pattern, we have to write all the control specific code.

- But in an MVT, the controller part is taken care of by the framework itself.

- Let us take an example to understand this difference.

# **M**odel-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT (contd…)**

- Assume that you have to display a list of books you have in a library, stored is a table named *books*. In MVC architecture, you have to write the code to fetch the list of books from the database, write the presentation layer (i.e HTML, CSS), map it with a URL and send it to the user.

- But if MVT is followed, you don't have to write any code related to fetching data from the database and mapping it with the URL. All these activities are handled by the framework itself. The only thing that you have to do is to tell the framework what data should be presented to the user (Books table). The framework will then create a view based on the data and send it to the user.

# **M**odel-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT (contd…)**

- The classic MVC pattern works by managing the state of an application. When a user performs an action or makes a request, an action in the Controller is called. The Controller then either tells Model to make changes and update the View or returns a View based on a Model. Hence we can say that the View is controlled by the Controller and Model.

- MVT takes a slightly different approach. When a user makes an HTTP request, the corresponding View performs a query on the Model and collects the result set from the Model. The View then fills the result in a template and sends it to the user.

# **M**odel-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT (contd…)**

- MVC helps to organize the core functions of your code neatly. It will be easy to make any changes to your app later as you will be able to identify which code does want.

# **M**odel-**V**iew-**T**emplate (MVT)

- **Lab Task for the week – 28 August 2022 to 03 September 2022**

- Compare and Contrast MVC & MVT.
- Django Application Structure