Unit:1

Vikash Rajpurohit

IOS

⊛ IOS Architecture

→ Architecture of IOS is a layered architecture.
→ There are four layers present here.
→ At the uppermost level IOS works as an intermediary between the underlying hardware and the apps you make.
→ Applications do not communicate to the hardware directly They communicate with each other using system Interface
→ These interfaces make it simple to write apps that work constantly on devices having various hardware abilities
→ Lower layer gives basic services on which all application depends.
→ Higher level layer gives sophisticated graphics and interface related services.
→ Every layer have a set of Framework which the developer use to construct the applications.

Cocoa Touch	View Hierarchy, Localization, Alerts, WebView, Map Kit, Camera.
Media Layer	Core Audio, OpenAL, Audio Mixing, Audio Recording, Video Playback
Core Services	Collections, Address Book, Core location, Net Services, Threading
Core OS.	Sockets, ~~Hardware~~ Power Managment, File System.

→ Lowest level → Core OS

→ Uppermost level → Cocoa Touch

↓

User interact directly with ~~Hardware~~ application

→ Core OS had low level features that most technology are build upon.

→ Core ~~OS~~ Services it provide various services.

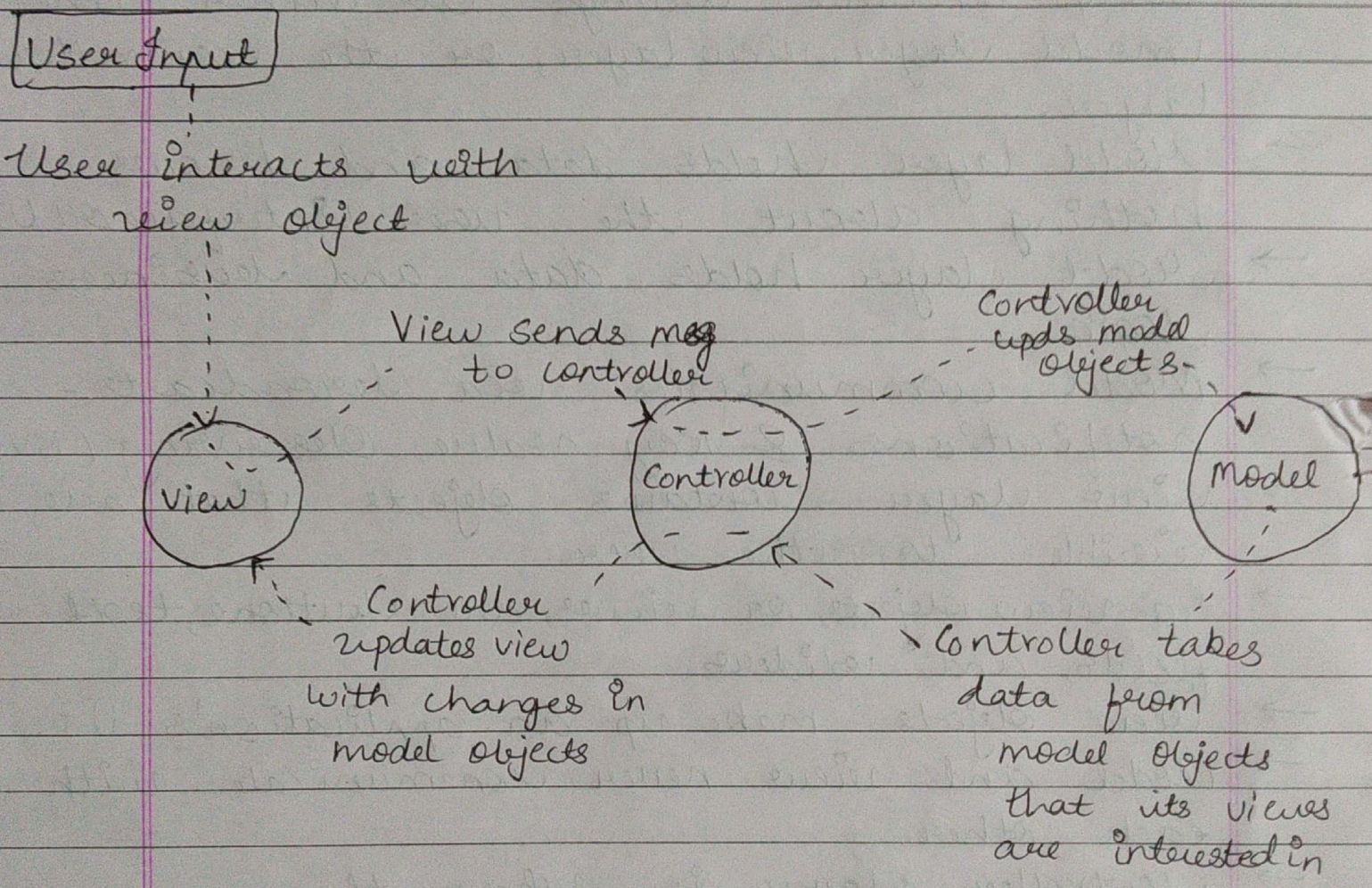
→ Media Layer it provide service related to media like audio, Video, Images etc.

⊛ Model - View - Controller

→ MVC is a design pattern used in IOS development.
→ Every instance belongs to either the model layer, view layer, or the controller layer.
→ Model layer holds data and knows nothing about the user interface or UI.
→ Model layer holds data and business logic.
→ Model communicates via broadcasts - notifications & Key value Observing (KVO)
→ View layer contains objects that are visible to the user.
→ Eg: view objects, or views, are buttons, text fields, and sliders.
→ View objects make up an application's UI.
→ Model and view never communicate with each other.
→ Controller layer is where the application is managed.
→ Controllers configure the views that the user sees and makes sure that the views and model objects stay synchronized.

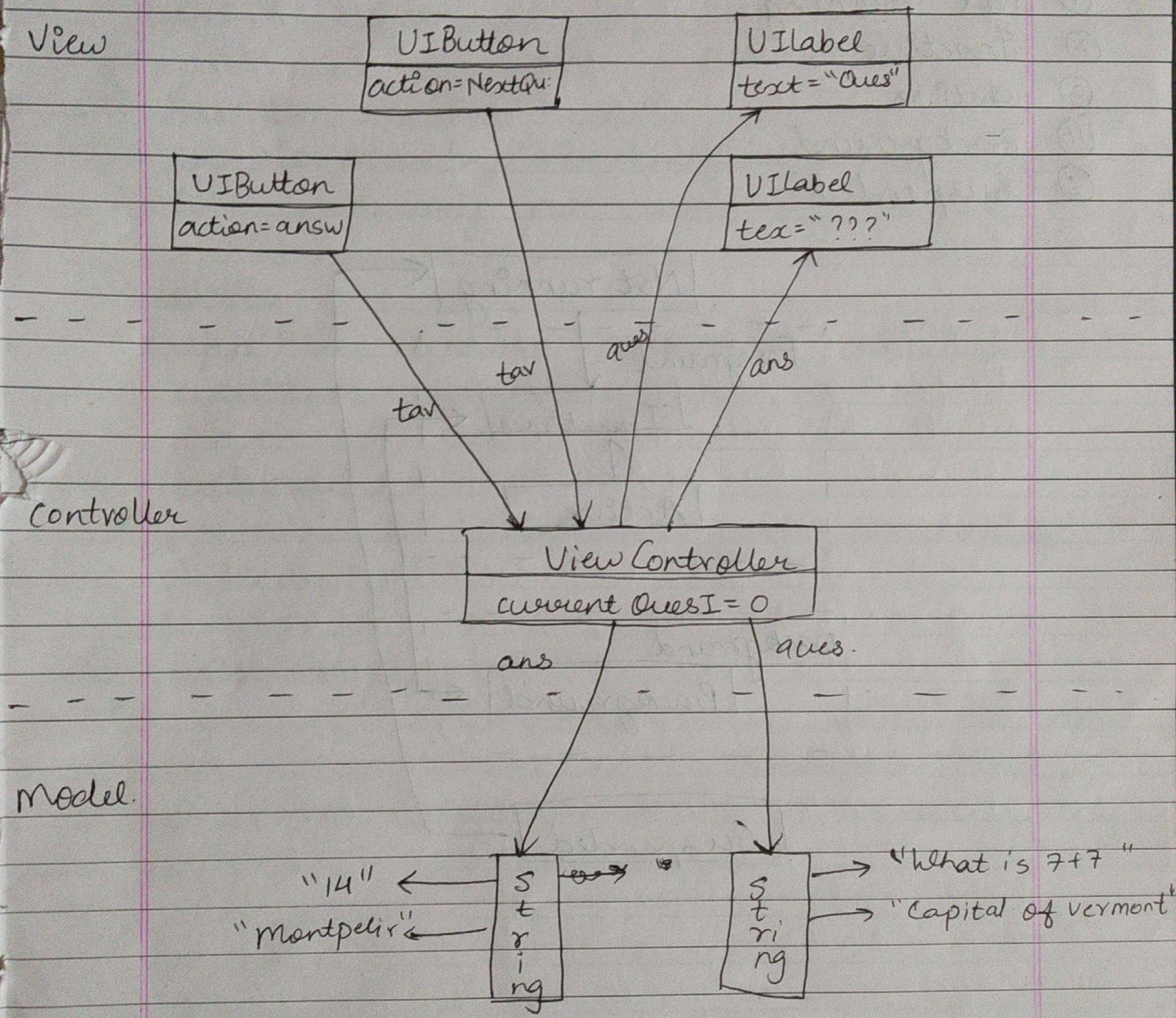
→ Controller can directly control Model and View.

→ View communicates with Controller via action, Outlet, Delegate, Data source.



MVC pattern

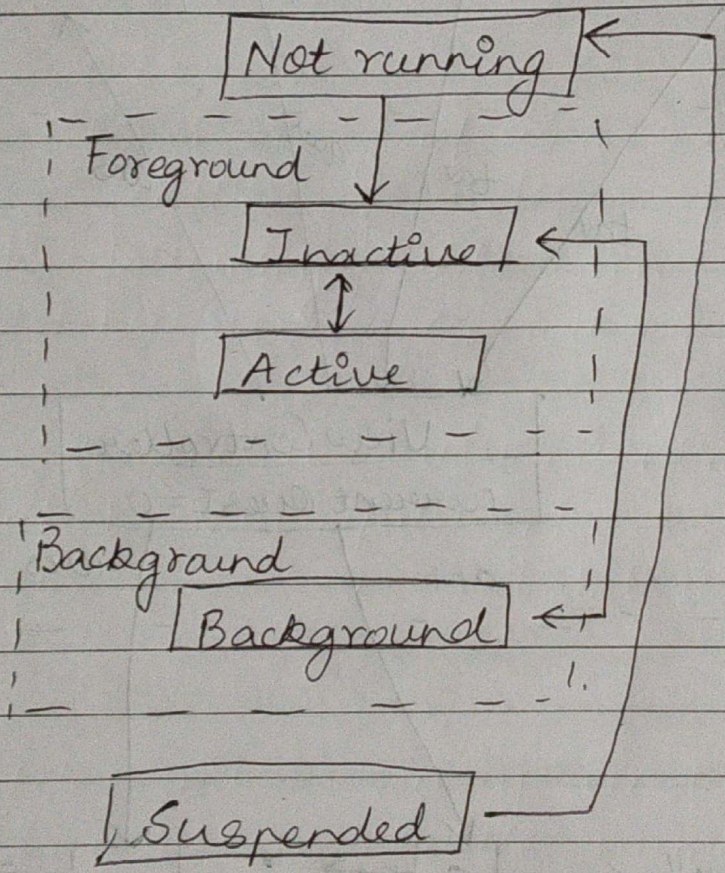
eg: Quiz application



VALENTINE
Page No.:
Date: / /

(*) Lifecycle states.

1. Not running
2. Inactive
3. Active
4. Background
5. Suspended



① Not running
→ app has not been launched or terminated by the system.

② Inactive
→ app is running in the foreground but is currently not receiving events.

③ Active
→ app is running and is receiving events. In foreground apps.

④ Background
→ app is in background and executing code.

⑤ Suspended
→ app is in background but is not executing code. App remains in memory but does not execute any code.

(*) Making connections.

→ A connection lets one object know where another object is in memory so that the two objects can communicate.

→ Two types of connections:

① Outlets

② Actions.

→ Outlets: it is a reference to an object.

→ Declaring outlets

Write the following code in class ViewController: UIViewController {}.

```
class ViewController: UIViewController {
    @IBOutlet var questionLabel: UILabel!
    @IBOutlet var answerLabel: UILabel!
}
```

→ This code gives every instance of ViewController an outlet named questionLabel and answerLabel.

→ ViewController can use each outlet to reference a particular UILabel object.

→ Outlet keyword tell Xcode that you will connect these outlet to label objects using Interface Builder.

→ Setting outlets
→ Control-drag (or right-click and drag) from View Controller in the document outline to the top level in the scene.
→ When the label is highlighted, release the mouse and keyboard; a black panel will appear. Select appropriate option to set the outlet.
→ Notice that you drag from the object with the outlet that you want to set to the object that you want that outlet point to.

→ Defining action methods.
→ When a UIButton is tapped, it calls a method on another object.
→ That object is called the target.
→ The method that is triggered is called the action.
→ This action is the name of the method that contains the code to be executed in response to the button being tapped.
Write following code in class.

```
class ViewController: UIViewController {
  @IBAction func showNextQuestion(_ sender:
  UIButton) { }
```

→ The @IBAction keyword tells Xcode that you will be making these connections in Interface Builder.

→ Setting targets and actions
→ To set an object's target, you control-drag from the object to its target.
→ When you release the mouse, the target is set, and a pop-up menu appears that lets you select an action.

→ You can check these connections in the connections inspector.

VALENTINE
Page No.:
Date: / /

(*) Code for creating a Quiz application.

→ Create View
→ Add connections

```
class ViewController : UIViewController {
  @IBOutlet var questionLabel: UILabel!
  @IBOutlet var answerLabel : UILabel!

let questions: [String] = [
    "what is 7+7",
    "What is the capital of Vermont?",
    "What is cognac made from?
]

let answers: [String] = [
    "14",
    "Montpelier",
    "Grapes"
]

var currentQuestionIndex: Int = 0

@IBAction fun showNextQuestion(_
sender: UIButton) {
currentQuestionIndex += 1
if (currentQuestionIndex == questions.count
{ currentQuestionIndex = 0 }
```

```
let question: String = questions[currentQuestionIndex]
questionLabel.text = question
answerLabel.text = "???"
}

@IBAction func showAnswer(_ sender: UIButton) {
  let answer: String = answer[currentQuestionIndex]
  answerLabel.text = answer
}

override func viewDidLoad(){
    super.viewDidLoad()
  questionLabel.text = questions[currentQuestionIndex]
}
}
```